Abstract Interpretation using Domain Theory

Flemming Nielson

Doctor of Philosophy

University of Edinburgh

1984

ABSTRACT

A framework is developed for describing and proving the correctness of certain data flow analyses. This is done by ascribing several semantics to the programming language studied. The standard semantics is the usual semantics and an approximating semantics describes a data flow analysis. A value in the latter semantics describes a set of values in the former and this is expressed using the framework of abstract interpretation pioneered by P. and R. Cousot. Their view of a programming language is limited because a program is viewed as a (kind of) flowchart and the main aim of this work therefore is to extend the framework to all programming languages that have a denotational semantics. This is accomplished except for "storable procedures". A secondary aim is to extend abstract interpretation to include certain aspects of termination.

The first aim is addressed by studying a metalanguage for denotational semantics. A collecting semantics is defined and it may be viewed as the most precise of all data flow analyses. Its formulation requires a study of (relational) powerdomains. It is then proved correct and "as precise as possible" with respect to the standard semantics. The development of abstract interpretation generalises the previous approaches. In particular, the tensor product is found to generalise the relational method just as cartesian product corresponds to the independent attribute method. It is studied how to pass between such methods and how functionals like conditional are likely to look.

To achieve the second aim another powerdomain is required. The development of abstract interpretation distinguishes between two partial orders: $\sqsubseteq$ is the usual partial order of denotational semantics and $\subseteq$ expresses the data flow analysis idea of "safe approximation". (It was $\sqsubseteq$ above.) An example is given that requires this framework. Another example studies a nondeterministic language and its usual (nondeterministic) denotational semantics. It is shown that the semantics may be viewed as a data flow analysis upon a deterministic language where oracles resolve the choice of paths. This view motivates the definition of a nondeterministic denotational semantics that distinguishes between "may diverge" and "may have to diverge".

PREFACE

My work upon combining abstract interpretation with denotational semantics started when I was a M.Sc. student and I learned much about data flow analysis from my supervisor N. Jones. As a Ph.D. student I learned much domain theory from my supervisor G. Plotkin and this cleared the way for a much more general combination. I would also like to thank A. Mycroft for sharing my interest in abstract interpretation.

This research was funded for two years and six months by The Danish Natural Science Research Council.

My wife already knows my gratitude to her.

DECLARATION

I hereby declare that this thesis has been composed by myself, that the work presented has not been presented for any university degree before and that the work is my own except as follows. Major techniques suggested by my supervisor are indicated by a reference to /PloPS/. Sections 5.3 to 5.5 contain material that builds on chapter 2 of /Myc81/; this material was previously published as /MyNi83/ and in its present form is mostly my work. A preliminary version of sections 5.6 and 5.7 has been published as /Nie83/.

CONTENTS

1 INTRODUCTION

The idea behind abstract interpretation, the subject matter of this thesis, is best explained with an example taken from /CoCo77b/. The text -1515×17 may be interpreted to denote computations on the abstract universe $\{(+),(-),(\overset{+}{-})\}$ where the semantics of arithmetic operators is defined by the rule of signs. The abstract computation

$$-1515\times17 \rightarrow -(+)\times(+) \rightarrow (-)\times(+) \rightarrow (-)$$

shows that -1515×17 is a negative number. Abstract interpretation is concerned with a particular underlying structure of the universe, in this example the sign. It gives a summary of some facets of the actual computations of a program. In general this summary is simple to obtain but inaccurate, e.g. $-1515+17 \rightarrow -(+)+(+) \rightarrow (-)+(+) \rightarrow (\overset{+}{-})$.

The results of abstract computations should be correct with respect to the semantics of the program. Usually this has been considered with respect to an operational semantics for flowcharts or occasionally a predicate transformer semantics. The purpose of this thesis is to develop abstract interpretation in the framework of denotational semantics /Sto77/ with its underlying domain theory. Denotational semantics is used partly because it allows considering programs in their source representation and partly because denotational semantics has been used to give semantics for most language features.

In the remainder of this chapter we shall

- overview a general setting where abstract interpretation may be encountered,

- informally review abstract interpretation,
- overview some earlier denotational formulations of properties of computations,
- describe the aims of the work reported and
- give an overview of the contents of the thesis.

The main body of previous work that is built upon is abstract interpretation (e.g. /CoCo77b/ and /CoCo79/) and domain theory (e.g. /SmP182/ and /PloLN/). It is helpful if the reader is acquainted with abstract interpretation but the overview in section 1.2 may be sufficient. The domain theory needed is reviewed in section 2.2 but this overview may be hard to follow if the reader is not already acquainted with "Scott's ideas". Some set theory (e.g. /Hal60/) is needed in chapter 5.

1.1 THE GENERAL SETTING

The motivation behind the subject matter of this thesis concerns the implementation of programming languages whether by compilation (e.g. ALGOL and PASCAL) or by interpretation (e.g. APL and SETL). Compilation may be viewed as consisting of lexical analysis, syntax analysis, generation of intermediate code, code optimization, generation of machine specific code and peephole optimization /AhU178/. The phase of interest for abstract interpretation is code optimization. Similar considerations apply for the interpretation of programs.

Code optimization may be viewed as consisting of many program transformations. The purpose of each is to improve the quality of the program with respect to some measure (e.g. running-time, program length, storage requirements) while preserving the meaning of the program. For an example consider the program

... y:=2 ... (no y's) ... x:=y+(1+1) ... (no x's) ... x:=0 ...

and the replacement of x:=y+(1+1) by x:=y+2. It is correct to do so because the two pieces of program have the same meaning. Many interesting transformations are not of this kind and rely on data flow analysis to provide information about the context in which the program transformation is to be performed so that the two pieces of program need not have identical semantics. This is illustrated by the following two examples taken from /Nie81b/.

The first transformation (constant folding /AhU178/) is to replace x:=y+(1+1) by x:=4. These two pieces of program have different semantics but the transformation is semantically correct because y will always have the value 2 before the assignment statement in question. The data flow analysis constant propagation /AhU178/ computes before each statement a (safe approximation to a) set of variable and value pairs, so that a given pair is included iff the variable has always that value at that point. This analysis is classified as a forward analysis /Hec77/ because it propagates information from left to right.

Another transformation (useless code elimination /AhU178/) is to remove x:=y+(1+1). The semantics of this piece of program does not equal that of the empty statement but the transformation is correct because the value of x is not subsequently used before x has been assigned a new value. The data flow analysis live variable analysis /AhU178/ computes after each statement (a safe approximation to) the set of variables that in some computation will be used before having been assigned a new value. The analysis is a backward analysis /Hec77/ because it propagates information from right to left.

The data flow information computed is a safe approximation to the desired set. The need for approximation arises because the problem of determining for a program whether an element is in the desired set at a given point, is in general not decidable. For live variable analysis the problem is recursively enumerable and for constant propagation is the complement of recursively enumerable. By safe is meant that the computed set should be smaller/larger than desired if this means fewer transformations seem allowed. For live variable analysis safe means larger and for constant propagation it means smaller.

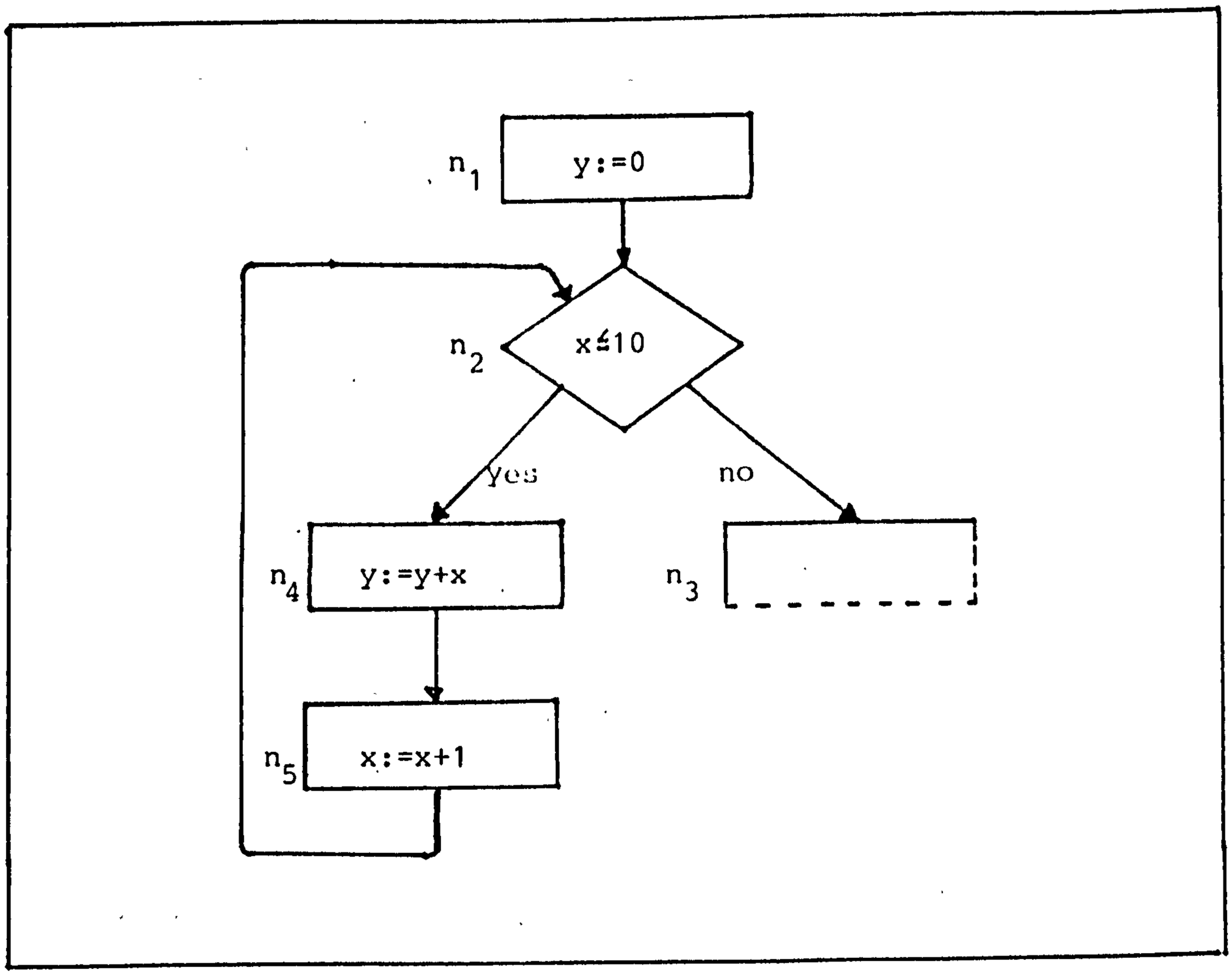
In code optimization a program is usually represented as a flowchart whose nodes contain sequences of very simple assignment statements. Most data flow analyses have been developed using this scheme (see /Hec77/ for an overview). However, it is often possible to perform much optimization at the source level (perhaps extending it with some primitives). An advantage of this is that the transformed program may be displayed in a way understandable to the programmer. For an indication of the potential efficiency gains one may note /BaSa74/, where it is estimated that 80% of the run-time dimension, type and value checks performed in a naive APL interpreter may be eliminated using data flow information. It therefore seems worthwhile to develop data flow analysis at this level. This has been done (/Hec77/,/Ros77/,/Wil81/,...) but like most data flow analyses the information obtained is not proved correct with respect to any formal semantics. Such a programme is possible using the ideas of abstract interpretation. Some data flow analyses that can be handled using abstract interpretation are constant propagation, type determination and information for array bound checking.

1.2 ABSTRACT INTERPRETATION

Data flow analysis specifies at each program point information about the context in which it appears. Often this information describes (a safe approximation to) the possible states at that point. Constant propagation is an example of this but live variable analysis is not. The important step of formally relating data flow information to sets of states was taken by Patrick and Radhia Cousot in their work on abstract interpretation /CoCo76,CoCo77a,CoCo77b, CoCo78,CoCo79,Cou81/. In data flow analysis much work has been directed towards efficient algorithms for performing the data flow analyses but this has played a small role in abstract interpretation and we shall ignore the issue in this thesis.

To explain abstract interpretation consider the following flowchart. Nodes are labelled by n_i and n_2 is a test node whereas the others are assignment nodes. We shall assume that the variables x and y take their values in the set $Z = \{...,-1,0,1,...\}$ of integers. The set of states may be taken to be Z^2. The semantics of an assignment node is given by a (total) function $f_i:Z^2 \rightarrow Z^2$. For the test node n_2 there is a predicate (i.e. a total function) $f_2:Z^2 \rightarrow \{tt,ff\}$. Here tt means true and ff means false.

A first step is to consider the collecting semantics (called the static semantics in /CoCo77b/). The aim is to specify for each node n_i the set $G_i \subseteq Z^2$ of states possible at entry to that node. It is assumed that G_1 is some given set G_{start} (often Z^2). The semantics of an assignment node is given by a function $g_i:\mathcal{P}(Z^2) \rightarrow \mathcal{P}(Z^2)$ where $\mathcal{P}(Z^2)$ is the powerset of Z^2. It is defined



by

$$g_i(Y) = \{f_i(u,v) \mid (u,v) \in Y\}$$

The test node n_2 gives rise to two such functions:

$$g_{2,yes}(Y) = \{(u,v) \mid (u,v) \in Y \wedge f_2(u,v) = tt\}$$

$$g_{2,no}(Y) = \{(u,v) \mid (u,v) \in Y \wedge f_2(u,v) = ff\}$$

Intuitively the G_i must satisfy the following equations:

$$G_1 = G_{start}$$

$$G_2 = g_1(G_1) \cup g_5(G_5)$$

$$G_3 = g_{2,no}(G_2) \qquad \text{(G-equations)}$$

$$G_4 = g_{2,yes}(G_2)$$

$$G_5 = g_4(G_4)$$

Another way to view these equations is that they define a function g from $(\wp(Z^2))^5$ to itself, i.e.

$$g(G_1,G_2,G_3,G_4,G_5) = (G_{start}, \ldots, g_4(G_4))$$

We shall see that this system of equations always has a least fixed point solution, i.e. a solution that is included in every other solution. This fixed point solution (written G_i) is of interest because G_i contains precisely those states that may arise at entry to n_i for some computation starting at n_1 with a state in G_{start} /CoCo77b, Cou81/. For the example flowchart $G_{start} = \{(9,9)\}$ gives $G_4 = \{(9,0),(10,9)\}$ and hence x may be 9 or 10 at entry to n_4.

A possible data flow analysis is the detection of signs. Let $I = \{-,0,+\}$ be a "degraded" version of the integers. The idea is to replace $\mathcal{P}(Z^2)$ by $\mathcal{P}(I^2)$ and so represent a set $Y = \{(-1,1),(2,-2),(3,-3)\}$ by $X = \{(-,+),(+,-)\}$. The meaning of $X \subseteq I^2$ is formally defined as $con(I) \subseteq Z^2$ where $con: \mathcal{P}(I^2) \to \mathcal{P}(Z^2)$ is a <u>concretization function</u>. If the function $r: Z \to I$ is defined by $r(-1)=-$, $r(0)=0$, $r(1)=+$ etc. we may define

$$con(X) = \{(u,v) \mid (r(u),r(v)) \in X\}$$

Similarly there is an <u>abstraction function</u> $abs: \mathcal{P}(Z^2) \to \mathcal{P}(I^2)$ defined by

$$abs(Y) = \{(r(u),r(v)) \mid (u,v) \in Y\}$$

It satisfies that abs(Y) is the smallest set X such that con(X) includes Y. This relationship between abs and con will be formalised by the definition of pairs of adjoined functions below and is central to the theory of abstract interpretation.

One may then formulate a new set of equations:

$$G'_1 = G'_{start}$$

$$G'_2 = g'_1(G'_1) \cup g'_5(G'_5)$$

$$G'_3 = g'_{2,no}(G'_2)$$
$$G'_4 = g'_{2,yes}(G'_2) \qquad \text{(G'-equations)}$$
$$G'_5 = g'_4(G'_4)$$

It is natural to define $g'_i = abs^{\bullet}g_i^{\bullet}con$ as this intuitively is the most precise account of g_i (and f_i) that can be given when considering subsets of I^2. We shall define $g'_{2,yes}$ and $g'_{2,no}$ similarly although these are frequently assumed to be identity functions, i.e. $g'_{2,yes}(X) = X = g'_{2,no}(X)$. For G'_{start} it is natural to use $abs(G_{start})$. We shall see that a least fixed point solution G'_i exists and that $con(G'_i) \supseteq G_i$. Because of the relation between G_i and the set of states possible before n_i we see that G'_i describes a superset of this set. Returning to the example we will get $G'_{start} = \{(+,+)\}$ and $G'_4 = \{(+,0),(+,+)\}$.

Not all data flow analyses use powersets. In /Kil73/ and /KaUl77/ a theory of monotone frameworks was developed where powersets are replaced by complete lattices of finite height (see below). Most data flow analyses fall within this framework and efficient solution methods have been developed (e.g. /GrWe76/). Abstract interpretation may be viewed as extending the theory with means for showing the specified solutions to be semantically correct (using concretization functions). To explain this and the theory for when least fixed points exist we need some lattice theoretic notions. More details can be found in /Cou81/.

For the overview of lattice theoretic concepts let $L = (L,\sqsubseteq)$ be a partially ordered set. An <u>upper bound</u> of a subset H of L is an element l of L such that $h \sqsubseteq l$ whenever h is an element of H. The

subset H has a (necessarily unique) least upper bound denoted $\bigsqcup H$ iff $\bigsqcup H$ is an upper bound such that $\bigsqcup H \sqsubseteq l$ for every upper bound l. Dually (i.e. using $\sqsupseteq$) there may be a greatest lower bound $\bigsqcap H$. If all least upper bounds exist we say L is a complete lattice and then all greatest lower bounds exist too because

$$\bigsqcap H = \bigsqcup\{l \mid \forall h \in H: l \sqsubseteq h\}$$

It is convenient to write $\bot = \bigsqcup\emptyset$, $\top = \bigsqcap\emptyset$, $x \sqcup y = \bigsqcup\{x,y\}$ and $x \sqcap y = \bigsqcap\{x,y\}$. An (increasing) chain is a sequence $(l_n)_n$ of elements of L that are indexed by the natural numbers $N = \{1,2,\ldots\}$ in such a way that $l_1 \sqsubseteq l_2 \sqsubseteq \ldots$. If no such chain has infinitely many distinct elements then L is said to be of finite height. The least upper bound $\bigsqcup\{l_n \mid n \in N\}$ of the chain $(l_n)_n$ is written $\bigsqcup_n l_n$. By $L^n = L \times \ldots \times L$ is meant the n-fold cartesian product of L with itself. The partial order is defined componentwise, i.e. $(l_1,\ldots,l_n) \sqsubseteq (l'_1,\ldots,l'_n)$ iff $l_1 \sqsubseteq l'_1$ and ... and $l_n \sqsubseteq l'_n$. It follows that L^n (for $n \geq 1$) is a complete lattice if L is.

A function $g: L \to M$ between partially ordered sets is monotonic iff $l \sqsubseteq l'$ implies $g(l) \sqsubseteq g(l')$. It is strict iff $g(\bot)$ is the least element of M whenever $\bot$ is the least element of L. It is completely additive iff

$$g(\bigsqcup H) = \bigsqcup\{g(h) \mid h \in H\}$$

whenever H is a subset of L that has a least upper bound. It is additive iff $g(l \sqcup l') = g(l) \sqcup g(l')$ whenever $l \sqcup l'$ exists. The partial order $\sqsubseteq$ on M may be extended pointwise to functions by defining $g \sqsubseteq g'$ iff $g(l) \sqsubseteq g'(l)$ holds for all elements l of L. A fixed point of $g: L \to L$ is an element l of L such that $g(l) = l$. It is the least

<u>fixed point</u> if $l \sqsubseteq l'$ holds for all fixed points l'. If L is a complete lattice and g is monotonic there always is a least fixed point and it is given by

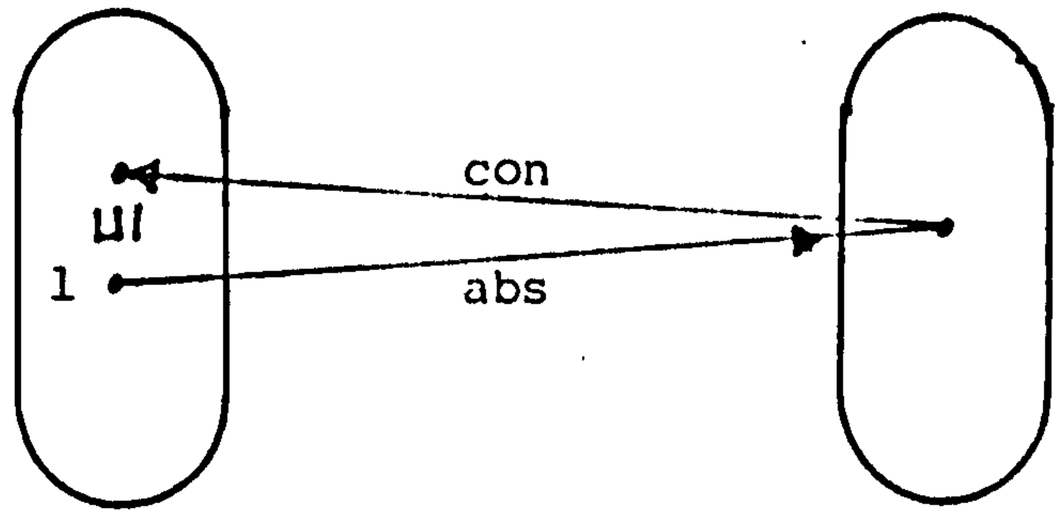
$$LFP(g) = \bigsqcap \{l \mid g(l) \sqsubseteq l\}$$

This result is due to /Tar55/ and may be proved as follows. If $g(l) \sqsubseteq l$ the monotonicity of g gives $g(LFP(g)) \sqsubseteq g(l) \sqsubseteq l$ and therefore $g(LFP(g)) \sqsubseteq LFP(g)$. Further, monotonicity gives $g(g(LFP(g))) \sqsubseteq g(LFP(g))$ and hence $LFP(g) \sqsubseteq g(LFP(g))$ follows from the definition of LFP(g).

Turning to abstract interpretation we now address the details that were omitted before. The formulation of the collecting semantics used the complete lattice $(\wp(Z^2), \subseteq)$ but this will not be essential below so we shall just assume a complete lattice $L = (L, \sqsubseteq)$. It was already said that the equations define a function g from L^n to L^n (with n=5). It is not hard to see that g is monononic and hence there is a least fixed point as stated. We shall write $(G_1, \ldots, G_n)$ for this least fixed point. In the detection of signs analysis there is a monotonic function g' from L'^n to L'^n. In the example L' is $(\wp(I^2), \subseteq)$ but we shall only need that L' is a complete lattice. Again g' has a least fixed point which is written $(G'_1, \ldots, G'_n)$.

It remains to relate the G'_i to the G_i. We already saw that this is to be done using a concretization function con:L'→L and an abstraction function abs:L→L'. It is apparent from the two examples given that the partial order represents the data flow analysis idea of "can safely be approximated by". It is therefore natural to assume that abs and con are monotonic for this just means that they preserve "safe approximation". We shall also assume that

$con(abs(l)) \sqsupseteq l$ is always the case. This condition may be illustrated by



and it merely says that abs(l) is a safe description of l. It is common also to assume that $abs(con(l')) \sqsubseteq l'$ holds for all l'. This implies that abs(l) is the least element that safely describes l, i.e. that

$$abs(l) = \bigsqcap\{l' \mid con(l') \sqsupseteq l\}$$

because whenever $con(l') \sqsupseteq l$ we get $l' \sqsupseteq abs(con(l')) \sqsupseteq abs(l)$. These conditions upon abs and con may be reformulated as

$$\forall l \in L: \forall l' \in L': \quad abs(l) \sqsubseteq l' \Leftrightarrow l \sqsubseteq con(l')$$

and (abs,con) is said to be a <u>pair of adjoined functions</u> whenever this is the case /CoCo79/. It is easy to see that this condition holds for the particular abs and con displayed earlier. The adjoinedness condition is further motivated and studied in /CoCo79/ and /Nie81a/. One property worth noting is that abs is completely additive (as is straightforward to show).

The desired relation between G_i' and G_i is that $G_i \sqsubseteq con(G_i')$, i.e. that G_i' is a safe description of G_i. Writing $con^n(l_1',\ldots,l_n') = (con(l_1'),\ldots,con(l_n'))$ this may be written $LFP(g) \sqsubseteq con^n(LFP(g'))$. To achieve this it suffices to show that $g \cdot con^n \sqsubseteq con^n \cdot g'$ for then one has

$$g(con^n(LFP(g'))) \sqsubseteq con^n(g'(LFP(g'))) = con^n(LFP(g'))$$

and by /Tar55/ the desired result follows. When (abs,con) is a pair of adjoined functions also (abs^n,con^n) is and $g \cdot con^n \sqsubseteq con^n \cdot g'$ is then equivalent to $abs^n \cdot g \cdot con^n \sqsubseteq g'$. Returning to the examples considered earlier it is straightforward to use the complete additivity of abs to show that equality holds and hence G'_i is a safe description of G_i. The function $abs^n \cdot g \cdot con^n$ is said to be <u>induced</u> from g /CoCo79/ and is of interest because it intuitively is the best representation of g that can be obtained using the approximate space L'.

Abstract interpretation is a quite general framework and we shall consider a few additional examples. The detection of signs analysis (as well as the collecting semantics) is a <u>relational method</u> /Jon81a/ because the use of $L' = \mathcal{P}(I^2)$ means that all combinations of values are considered. An alternative might be to use $L'' = (\mathcal{P}(I))^2$ which gives an <u>independent attribute method</u> where combinations of values cannot be explicitly considered. The relation between L' and L" may be defined by the following concretization and abstraction functions

$$con'(Y_1,Y_2) = Y_1 \times Y_2$$

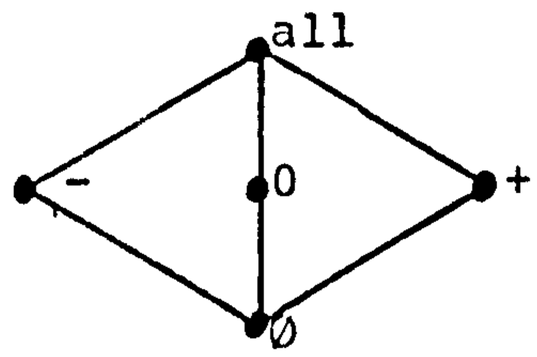
$$abs'(Y) = (\{u \mid \exists v: (u,v) \in Y\} , \{v \mid \exists u: (u,v) \in Y\})$$

As before a system of equations may be defined where $G''_{start} = abs'(G'_{start})$ and $g''_i = abs' \cdot g'_i \cdot con'$. This gives a monotonic function g" from $(L")^n$ to $(L")^n$ and it is induced from g'. It follows that the least fixed point satisfies $G'_i \sqsubseteq con'(G''_i)$ and hence

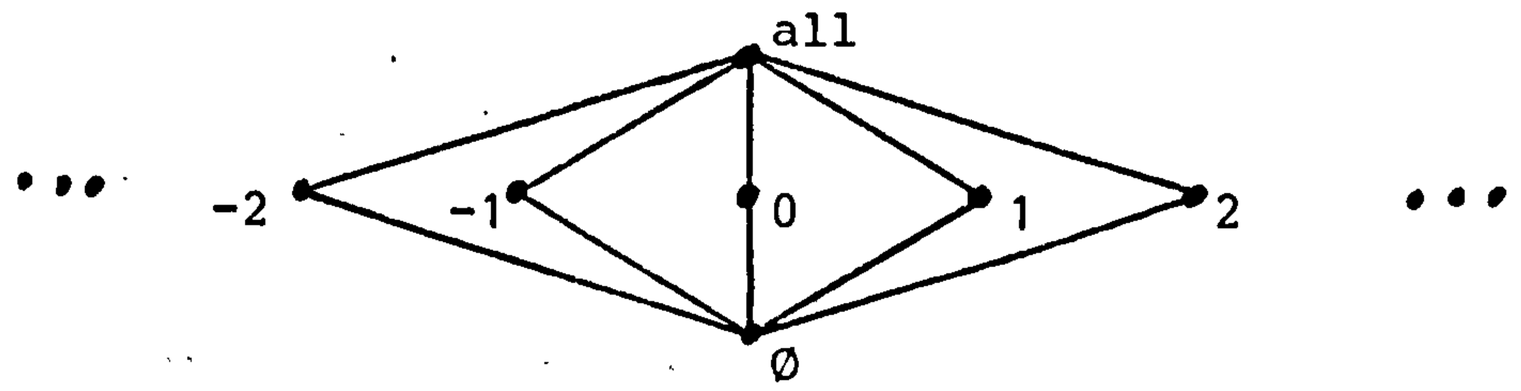
$$G_i \sqsubseteq (con \cdot con')(G''_i)$$

so that G''_i is a safe description of G_i.

One can move closer to the example given in the introduction by replacing $\mathcal{P}(I)$ with



Here the elements are ∅,-,0,+,all and the partial order has $x \sqsubseteq y$ iff there is a non-descending path from x to y. One way to say that the above space is more approximate than $\mathcal{P}(I)$ is to note that more subsets have the same least upper bounds. So the least upper bound operation may be viewed as expressing the "degree of approximation" just as the partial order expresses "can safely be approximated by". Another example where many sets have the same least upper bounds are in <u>constant propagation</u> where $\mathcal{P}(Z)$ is replaced by



with the concretization function being the obvious one.

So far it has been assumed that the least fixed point is the desired solution. It is usually called the <u>MFP</u> solution (for maximal fixed point) because originally $\sqsupseteq$ was used where we have used $\sqsubseteq$. Another solution is the <u>MOP</u> solution (for meet over paths). If p is a path from the entrance of node n_i to the entrance of n_j the composite effect of the $\bar{g}_k$'s (expressing the effect of the nodes) is written $\bar{g}_p$, e.g.

$$\bar{g}_{n_1 n_2 n_4 n_5} = \bar{g}_4 \cdot \bar{g}_{2,yes} \cdot \bar{g}_1$$

The MOP solution specifies that

$$\bar{M}_i = \bigsqcup\{\bar{g}_p(\bar{M}_{start}) \mid p \text{ is a possibly cyclic path from } n_1 \text{ to } n_i\}$$

is the desired information at entrance to n_i. In the following we shall make use of the fact that there is no nonempty path that ends in n_1. If $\bar{M}_{start} = \bar{G}_{start}$ one can prove that $\bar{M}_i \sqsubseteq \bar{G}_i$ by induction on the maximal length of paths considered /KaUl77,CoCo77b/. If each $\bar{g}_k$ is completely additive one can show that $(M_1,\ldots,M_n)$ is a fixed point of the corresponding system of equations and hence $\bar{M}_i = \bar{G}_i$. This is the case for the collecting semantics (where $\bar{g}_k = g_k$). If not all $\bar{g}_k$ are completely additive the solutions may be different and this is the case for constant propagation (for a suitable flowchart /KaUl77/).

It is usually agreed that the MOP solution is the ideally desired solution rather than the MFP solution. Also it is quite easy to show that if $\bar{M}_{start} \sqsubseteq con(\bar{M}'_{start})$ and $\bar{g}_k \cdot con \sqsubseteq con \cdot \bar{g}'_k$ then the respective MOP solutions satisfy $\bar{M}_i \sqsubseteq con(\bar{M}'_i)$. Despite this the MOP solution is not usually computed. This is because there are complete lattices of finite height (e.g. as in constant propagation) such that computation of the MOP solution corresponds to solving the halting problem /KaUl77/. By contrast the MFP solution is decidable if the complete lattice has finite height. The idea is that finite height guarantees that $g^n(\perp) = g(g^n(\perp))$ for some n and $g^n(\perp)$ then equals LFP(g). We omit the details required for a formal statement of this.

1.3 DENOTATIONAL FORMULATIONS OF PROGRAM PROPERTIES

Several papers give denotational formulations of properties of program computations. Some papers that have influenced the present work are surveyed below. A further reference is /Ple81/ that considers storage sharing in SCHEME (a dialect of LISP).

In /Don78/ an imperative language with while-loops is given a continuation-style standard semantics using locations. Then a non-standard semantics defining properties of computations is introduced. Where the standard semantics uses a flat domain of integers ($Z_\perp$ with elements $Z \cup \{\perp\}$ and $x \sqsubseteq y$ if $x=\perp$ or $x=y$) the non-standard semantics uses a complete lattice L having as elements certain nonempty subsets of $Z_\perp$. It is assumed that all elements of L are left-closed (i.e. if $x \sqsubseteq y$ and y is a member so is x) and a further condition that we shall ignore. All sets $\{\perp, x\}$ for $x \in Z_\perp$ are elements of L and the partial order is set-containment $\subseteq$. Where the standard semantics uses a domain expression $Z_\perp \times \ldots$ the non-standard one uses $L \times \ldots$ and a function $f: Z_\perp \times \ldots \rightarrow Z_\perp \times \ldots$ becomes $g: L \times \ldots \rightarrow L \times \ldots$. In other words an "independent attribute method" is used. This is complemented by a formal proof that $g(S) \supseteq \{f(x) \mid x \in S\}$ but equality is not considered; we shall see in section 3.3 that it will not hold in general. The idea in how to define L builds on /WaSh77/ (see section 4.1) which shares some ideas with abstract interpretation but is not as "developed".

A possible critique of such an approach is that one may only deduce the properties holding at the end of the program and not at other program points. The latter is achieved in /Don81/ for a similar language. A standard semantics is modified to produce a

list of the intermediate values arising at various program points. Then non-standard semantics are defined that map program points to properties holding there. For the four data flow analyses considered in the paper it is proved that the non-standard semantics includes descriptions of all the intermediate values produced. The notation used is that of the Semantics Implementation System (SIS /Mos79/). This may be the reason for why the formulation is further removed than /Don78/ from the monotone frameworks or abstract interpretation approaches. The SIS system will, however, loop upon iterative programs (as is explained in chapter 6) so the analyses cannot be automatically executed /Ple81/. The solution specified is not related to the MOP and MFP solutions of the traditional approach.

In /Nie82/ the framework of abstract interpretation is given a denotational formulation for a language similar to the previous. First the semantics is given using a store semantics (see /MiSt76/). It is then modified so as to map program points to the set of states possible there. Thirdly, a "static semantics" is obtained by modifying the previous semantics to accept sets of states as input. It is proved that with input S the states specified at some point are the union over the elements of S of those the previous semantics specified there. The fourth semantics replaces sets of states by complete lattices of description elements. These complete lattices are related to sets of states using pairs of adjoined functions (in fact a weaker condition) and the latter semantics is proved correct with respect to the "static semantics" (i.e. it specifies a safe approximation). All these semantics are in continuation style and it is shown that the abstract information specified is the MOP solution. It is also possible to obtain the MFP solution.

A companion paper /Nie81b/ shows that the data flow information can be used to assert the correctness of the program transformation method of constant folding (and to some extent also useless code elimination). In these papers the denotational formulation of abstract interpretation is mainly viewed as a convenient step in proving the correctness of program transformations depending on data flow information. It is not discussed whether it would be feasible to execute the denotational formulations directly (e.g. on a modified SIS system).

An application of abstract interpretation to the optimization of applicative programs is contained in /Myc81/. Further an attempt is made at extending abstract interpretation to include certain aspects of termination. The attempt is of rather limited applicability, however, and the main application is incorrect. The motivation for sections 5.3 to 5.5 of this thesis therefore was to present a correct and slightly more general development.

1.4 AIMS OF THE THESIS AND OVERVIEW

The denotational approaches to formulating abstract interpretation (or just data flow analyses) have only considered toy languages. This gives rise to two shortcomings:

a) it is not completely specified how an ordinary denotational semantics is changed to one that performs abstract interpretation,

b) it is unclear what language features in some realistic language might give rise to difficulties that have not been solved for some of the previous toy languages.

Further,

c) formulations generally use an "independent attribute method" even in cases where a "relational method" poses no technical problems.

The operational formulations of abstract interpretation also has shortcoming b). Shortcoming a) occurs in a slightly weaker form because some generality is obtained by considering arbitrary flow-charts but it is not clear that the flowchart view encompasses all language features. Shortcoming c) does not surface in the flowchart view but is still implicit.

A further shortcoming of these approaches (except in a crude way parts of /CoCo79/ and /CoCo78/) is that

d) no notion of termination is considered

This is a problem mostly ignored in data flow analysis but when considering ambitious program transformations (like in-line expansion of procedures) this seems to be needed.

The development in chapters 2, 3 and 4 is concerned with a), b) and c) above. First, the use of denotational semantics gives the potential for remedying a) because denotational semantics has been used to give the semantics of most realistic languages. To fully achieve a) a <u>metalanguage</u> is defined (in section 2.1) and the development works for any language whose semantics can be defined in that metalanguage. The types are classified into two levels and this gives information about which types in the collecting semantics should be powersets (in general not all types should). There are some syntactical limitations in the metalanguage that requires further

research to overcome. This means that b) is achieved because the language features that cannot be treated (e.g. procedures as parameters) are those whose semantics uses features not included in the metalanguage. Whether c) is achieved or not depends on generalizing the notion of "relational method". If this is taken to be "as in the collecting semantics" then some progress is made. In particular, the study of the tensor product (in section 3.2) seems to generalize the relational method of /Jon81a/ from powersets to a large class of complete lattices (as is discussed in section 4.5).

Some simplifying decisions have been made in order to allow the development to concentrate upon a), b) and c). An important one is not to consider program points. This means that the development in a sense is closer to /Don78/ than to /Nie82/. It is hoped that program points can be introduced much as in /Nie82/. Once this is done program transformations and implementation of the analyses may be considered. Another important decision is only to treat "domain-independent" issues of abstract interpretation. An example "domain-independent" issues is expected definitions of functionals like conditional. But examples of how to choose an approximation lattice for some particular purpose (e.g. type checking) will not be given as these can be found elsewhere.

Chapter 5 is concerned with d), the problem of expressing termination. The mathematics is more complicated so it is only done at the level of toy languages. The motivation for this work is to bring more data flow analyses within the world of abstract interpretation. One application is to validate a data flow analysis that has been found useful for detecting when call-by-name can be replaced

by call-by-value /Myc81/. Another, more philosophical, application is to show that nondeterminism is nothing but a particular data flow analysis. A more detailed overview of each chapter is given below.

Overview of chapter 2

The semantic metalanguage is defined in section 2.1. It consists of syntax for types (domains) and expressions (elements). The types are divided into two nearly identical levels. The level of a type is used to determine (in the collecting semantics) whether the domain or its powerdomain (roughly powerset) should be used. The levels might be viewed as separating what is sometimes called static and dynamic semantics. The metalanguage has one major limitation namely that functions are not "first class citizens".

The basic domain theory is reviewed in section 2.2. In particular domain constructors (like cartesian product) are defined and it is shown how to solve (possibly nested) recursive domain equations. Categorical concepts (mainly category, functor and limiting cone) are found useful for stating definitions and results. (These concepts are all explained in section 2.2.) Much of the material is quite standard so many facts are stated with no or little proof. The proofs given generally use order theoretic methods (as in /PloLN/ and /SmPl82/) rather than categorical methods (as in /LeSm81/). In particular, nested recursive domain equations is handled by considering certain functors over a category whose morphisms are pairs of functions.

The metalanguage will be interpreted in many different ways (one for each kind of semantics considered). It is therefore helpful to

define the notion of an interpretation to specify those aspects that are not fixed throughout the thesis. This is done in section 2.3 and the semantics of the metalanguage is then given relative to an interpretation. The standard semantics /MiSt76/ (i.e. the ordinary denotational semantics) is then defined by specifying the standard interpretation S.

The metalanguage has a "flavour" that will be useful later but differs from e.g. /Sto77/ (e.g. the functionality of conditional). So to show the "expressiveness" of the metalanguage section 2.4 gives examples of its use. The major example is to define the semantics of an imperative language with procedures (like SMALL in /Gor79/). The other example is a simple applicative language (recursion equation schemes).

Overview of chapter 3

The first step towards formulating abstract interpretation is to define the analogue of the collecting semantics of /CoCo77b/. (It was called the "static semantics" in /CoCo77b/ but this conflicts with the terminology of static versus dynamic semantics. Unfortunately the terminology "collecting semantics" was used in /Nie82/ to mean something slightly different.) Previously the powerset $\mathcal{P}(...)$ has been used but for the denotational formulation some powerdomain /Plo76,Smy78/ seems to be needed. Three candidates are considered in section 3.1 and the relational (or lower) powerdomain $\mathcal{P}_R(...)$ is chosen because $\mathcal{P}_R(Z_\perp)$ essentially is $\mathcal{P}(Z)$. The section then covers the necessary theory about this powerdomain.

Section 3.2 defines and studies the tensor product L⊗M of certain complete lattices L and M. The tensor product is of interest because $\mathcal{P}_R(D \times E)$ essentially is $\mathcal{P}_R(D) \otimes \mathcal{P}_R(E)$. The practical use of it is further descussed in section 4.5 and it is claimed that the tensor product allows for a generalization (from powersets to complete lattices) of the relational method of /Jon81a/; the independent attribute method then amounts to use of cartesian product. The treatment of the tensor product in section 3.2 includes giving different characterisations of it (one as a function space) and a picture of a tensor product of non-powerdomains.

The collecting interpretation is defined in section 3.3. The idea is that a function $f: Z_\perp \times Z_\perp \to Z_\perp$ becomes $g: \mathcal{P}_R(Z_\perp) \otimes \mathcal{P}_R(Z_\perp) \to \mathcal{P}_R(Z_\perp)$. The major theorem is to show that this is done in such a way that g is the "pointwise" application of f. Intuitively this means that $g(Y) = \{f(y) \mid y \in Y\}$. This result requires the use of several techniques, most notably the use of predicates defined on domains given as solutions to recursive domain equations. Luckily some of the results are proved in sufficient generality that they can be reused in chapter 4.

Overview of chapter 4

The development of abstract interpretation builds on the notion of a pair of adjoined (abstraction and concretization) functions. To stay within the usual (continuous) domain theory some additional assumptions are needed and these are formulated in section 4.1. Further, two additional properties upon the abstraction functions are studied. One amounts to when it could have been defined by a

"representation function". The other is that so-called "essential" elements are preserved. This condition is found of use in section 4.4. It is argued that this property formalises a vague intuition about when one approximation space is more natural than another.

The main development of abstract interpretation is performed in section 4.2. The idea is that a data flow analysis is specified as an interpretation with certain properties; such interpretations are called approximating interpretations. The collecting interpretation should be thought of as an approximating interpretation that is approximated by all others. First, the notion of when one approximating interpretation is safely approximated by another is defined using a family of adjoined abstraction and concretization functions (indexed by the types in one of the levels). It is shown that this condition is sufficient and necessary for a similar condition ($\trianglelefteq_{con,t}$) to hold between the semantics of expressions given by the two approximating interpretations. When g and h correspond to nodes in the flowchart the condition $g \trianglelefteq_{con,t} h$ is nothing but the condition

$$g \cdot con \sqsubseteq con \cdot h$$

met earlier. Secondly, it is possible to specify an induced interpretation (generalizing the induced predicate transformers of /CoCo79/). This requires a given approximating interpretation, a family of more approximate spaces and a family of abstraction and concretization functions. It gives an approximating interpretation that uses the approximate spaces and is a safe approximation to the given interpretation. In a certain sense it is as precise as possible.

Section 4.3 considers how to define the abstraction and concretization functions by structural induction upon the indexing types. This was not of importance above but is in keeping with the principle of compositionality of denotational semantics. This is complicated by the possibility that the two data flow analyses use different "methods", e.g. one uses a relational method where the other uses an independent attribute method. The solution adopted is to specify the "change of method" using the categorical notion of a natural transformation. Several examples of such transformations are studied.

For some purposes an induced interpretation may be viewed as being too precise. Composition is an example of this. One may expect the definition of "composition" to be functional composition. Even if this is the case in some given approximating interpretation it needs not hold in the induced interpretation. It is, however, safe to use functional composition instead. Many similar examples can be given and these are studied in section 4.4. For some of these it will be required that the abstraction functions preserve "essential" elements. Finally, section 4.5 goes further into the discussion of why use of the tensor product gives a relational method.

Overview of chapter 5

In the beginning of chapter 5 it is argued that for some purposes a development based on the relational powerdomain is unsuitable. The problem is that it is impossible to express (in the collecting semantics) that some expression will terminate. To be able to do so

it is necessary to use another powerdomain and its theory is covered in section 5.2. It is sufficient for the purposes of this chapter but is not as general as might be desired (often the powerdomain is not a "domain"). In this development it is no longer valid to assume continuity and a "domain theory" based on monotonicity is covered in section 5.1 (but not for recursive domain equations).

Sections 5.3 to 5.5 consider a simple applicative language. First its standard and collecting semantics are defined and related. Because of the richness of the powerdomain the collecting semantics is a more faithful representation of the standard semantics. Secondly, it is argued that the powerdomain and all approximation spaces should be equipped with two partial orders. One represents "safe approximation" and the other corresponds to the usual partial order that is used in domain theory to define least fixed points. Chapters 2 to 4 may be viewed as a study of the special case where they are equal! The intuitive difference between these partial orders was already stated in /Nie81a/ and /Nie82/ and an attempt at formalising it was made in /Myc81/ (but using the wrong powerdomain). For such spaces a framework of abstract interpretation is developed. Thirdly, the applications show the correctness of a data flow analysis that was used in /Myc81/ to detect when call-by-name can be replaced by call-by-value.

Sections 5.6 and 5.7 study a simple nondeterministic language. The major aim is to show that the nondeterministic semantics of a program c is nothingbut a certain abstract interpretation of a deterministic program c'. The idea is to replace every occurrence

$$c_1 \underline{\text{or}} c_2$$

of a nondeterministic choice in c by

$$\underline{\text{if}}\ \text{hd}(o)\ \underline{\text{then}}\ o:=\text{tl}(o);c_1\ \underline{\text{else}}\ o:=\text{tl}(o);c_2$$

in order to get c'. Here o is an oracle /Mil75/. It is used to resolve which choice should be made and is modified so it can be used to resolve the next choice. The idea is that the semantics of c equals an abstract interpretation of c' where all oracles are assumed possible. To perform this development the powerdomain of section 5.2 seems needed and abstract interpretation is extended somewhat. It is then a natural data flow analysis question to investigate what happens if not all oracles are possible. Under some reasonable assumptions the amount of potential difference is shown to be bounded by the nondeterministic semantics and a modified semantics that distinguishes between "may diverge" and "may have to diverge".

2 STANDARD SEMANTICS

This chapter develops the framework of denotational semantics in which the formulation of abstract interpretation will be performed in chapter 4. The syntactic aspects of the semantic metalanguage are given in section 2.1 and the two-level type structure is explained. Applications of the metalanguage for defining semantics for example languages are given in section 2.4. The semantic aspects of the metalanguage are treated in section 2.3 and central to this is the definition of an interpretation. This is exemplified by defining the standard interpretation. The domain theory needed for this is developed/reviewed in section 2.2.

2.1 SYNTAX OF THE METALANGUAGE

The metalanguage has notation for defining types, which are to denote domains, and for defining expressions, which are to denote elements of domains. We begin with a study of the types and their two-level nature. Later the expressions are considered.

THE TYPES

The two levels of types are called the top-level and the bottom-level. We use the metavariable t for the former and gt for the latter. The syntax for the two levels is rather similar so underlining is used to disambiguate:

$$t ::= A_i \mid t_1 \times \ldots \times t_k \mid t_1 * \ldots * t_k \mid t_1 + \ldots + t_k \mid t_\perp \mid t_1 \to t_2$$
$$\mid \mathrm{rec}X_i.t \mid X_i \mid ft$$

$$ft ::= gt_1 \underline{\to} gt_2$$

$$gt ::= \underline{A}_i \mid gt_1 \underline{\times} \ldots \underline{\times} gt_k \mid gt_1 \underline{*} \ldots \underline{*} gt_k \mid gt_1 \underline{+} \ldots \underline{+} gt_k \mid gt_{\underline{\perp}}$$
$$\mid \underline{\mathrm{rec}}X_i.gt \mid X_i$$

The top-level types should be rather familiar. The A_i are unspecified basic types but we shall assume that the truth values T ... are included. Types are combined using k-ary type constructors: $\times$ is cartesian product, $*$ is smash product and + is coalesced sum. These are defined in section 2.2 and we shall assume that $k \geq 2$. To be precise we should be explicit about the value of k, e.g. have $\times^2$, $\times^3$ etc., but this would make the notation heavier and detract from readability. A new least element may be added by lifting, as in $t_\perp$, and function spaces may be defined, as in $t_1 \to t_2$. Recursive types are available via $\mathrm{rec}X_i.t$ and the possibility of using X_i in t. An example is recX.N+(N*X) that is the type of nonempty lists of integers (assuming N is the type of integers). This notation allows for nested use of recursive types. We shall postpone the discussion of ft (and gt).

For a type expression t and a finite set V of variables X_i, the static well-formedness condition $V \vdash t$ is defined by structural induction on t as indicated below. Among other things it says that the free variables of t are included in V.

t	A_i	$t_1 \ \ldots \ t_k$	...	recX.t	X	$gt_1 \underline{\to} gt_2$
$V \vdash t$	tt	$\bigwedge_{i=1}^{k} V \vdash t_i$	...	$V \cup \{X\} \vdash t$	$X \in V$	$\bigwedge_{i=1}^{2} \emptyset \vdash gt_i$

The use of the empty set $\emptyset$ in the rule for $V \vdash gt_1 \underline{\to} gt_2$ will be explained later. Type expressions t such that $\emptyset \vdash t$ will be of

special interest and are said to be closed.

It seems clear that the top-level types suffice for most imperative languages so it needs to be explained why a bottom-level is introduced. Consider command continuations /Sto77/ (of type C) for a language with states (of type S). In a continuation style semantics the meaning of a command is a continuation transformer (of type G). The definitions that are to be used are $G = C \rightarrow C$ in the top-level and $C = \underline{S} \underline{\rightarrow} \underline{S}$ in the bottom-level. For a standard semantics /MiSt76/ this distinction is superfluous so we shall consider a non-standard semantics. In the collecting semantics it should be clear that elements of C are to be functions from $\mathcal{P}(\underline{S})$ to $\mathcal{P}(\underline{S})$. Further, elements of G are still to be functions from C to C, not $\mathcal{P}(C)$ to $\mathcal{P}(C)$, as abstract interpretation is only concerned with the "state transformation" aspects and does not change a continuation style semantis to something else /Nie82/. So the use of the two levels is a welcome guide in the development of chapters 3 and 4.

Remark. One may ponder whether the use of two levels might be useful for other purposes than abstract interpretaion. An example that seems to show this is the case is code generation /MiSt76/. Here C is to be thought of as the domain of code (for state transformations) while G is still a domain of functions from code to code. In effect $\rightarrow$ is to be interpreted as a function space construction whereas $\underline{\rightarrow}$ is to be interpreted as textual representation for functions. Perhaps one may view the top-level as the place to express what is sometimes called static semantics and the bottom-level as the place to express dynamic semantics. In

particular it will emerge (later in this section and in section 2.4) that mutually recursive function definitions will be "solved" in the top-level. ///

The bottom-level types are much like the top-level types. In particular the unspecified base types $\underline{A}_i$ are assumed to include the truth-values $\underline{T}$. Domain constructors will be interpreted differently in various semantics and are underlined so they can be distinguished from those in the top-level. As an example $\underline{+}$ will be + in the standard semantics but in the collecting semantics it will be $\times$ because $\wp(\underline{A}+\underline{B})$ "is" $\wp(\underline{A})\times\wp(\underline{B})$ and not $\wp(\underline{A})+\wp(\underline{B})$. It might be of practical use in some semantics to have several versions of the domain constructors (e.g. $\underline{+}$, $\underline{\underline{+}}$, ...) so that they could be interpreted differently later and this should not cause any problems. Finally, the static well-formedness condition $V \vdash gt$ is defined much as before.

The main difference between the two levels is that the bottom-level has no function-spaces, i.e. we do not have gt ::= ft. Amending this would cause problems in chapter 4 that are not immediate to solve (see chapter 6). It is a consequence that e.g. procedures as parameters and "storable procedures" cannot be treated. This limits the generality of working at the metalanguage level but it is desirable that limitations in the theory are brought out at the syntactic level!

The interplay between the two levels is somewhat restricted. Using t ::= ft the bottom-level types may be included among the top-level types. This is useful e.g. for the function get defined in section 2.4. The functionality of get is $L \to \underline{S \to V \times S}$ which means that

get accepts a location and gives a "state transformation" from states to values and states. In the collecting semantics this becomes $L \to \mathcal{P}(\underline{S}) \to \mathcal{P}(\underline{V*S})$ so the idea is that identifiers always denote single locations. If this is not the case the metalanguage may be unsuitable because t ::= gt is not allowed. For the purposes of this thesis this is not felt too restrictive. It is in accord with the spirit of abstract interpretation where "state transformations" (i.e. ft-types) are of prime interest. Further, if t ::= gt is allowed then the close connection between standard semantics and the collecting semantics that is proved in the next chapter is likely to fail (see the discussion after theorem 3.3:14).

There is no mechanism for binding a top-level type into a bottom-level type. The need for doing so does not arise in this thesis. Also the possibility of viewing ft-types as "code" may suggest that this would not always be meaningful. Consequently the bottom-level is in a sense "below" the top-level. Finally, it should be explained why $V \vdash gt_1 \to gt_2$ iff $\emptyset \vdash gt_1$ and $\emptyset \vdash gt_2$. This enforces that no domain variable ranging over a bottom-level type (i.e. gt-type) can be bound (by rec) in the top-level and this simplifies part of the development. Another way to enforce this might be to distinguish between bottom-level and top-level domain variables and assume they can only be bound by a rec operator of the appropriate level.

Remark. Since a development of abstract interpretation involves making operational ideas denotational it may be well to ask whether the type constructors are natural from an operational point of view. One idea might be that functions in the top-level should be total functions between domains and that cartesian product corresponds to

tupling. Further, one might believe that functions in the bottom-level should be partial functions between predomains (domains possibly without $\perp$) and that cartesian product corresponds to tupling. We shall use total functions between domains but by using the smash product much the same effect is obtained. This suggests that smash product in the top-level and cartesian product in the bottom-level are not "natural". In fact these two type constructors give technical problems later (in particular in sections 4.2 and 4.4).

///

THE EXPRESSIONS

Next the syntax of expressions is defined. Contrary to the types it is possible to use "top-level notation" inside "bottom-level notation" and vice versa. The definition below annotates the various pieces of syntax with the corresponding domain constructors and suggests the intended meaning. Further hints are presented afterwards and the formal semantics is given in section 2.3.

e ::= $(e_1,\ldots,e_k)$ | $e'\downarrow i$	(for $\times$: tupling, indexing)
| $(*e_1,\ldots,e_k*)$ | $e'\downarrow i$	(for $\otimes$: tupling, indexing)
| in_i e' | is_i e' | out_i e'	(for +: inject,test,extract)
| up e' | def e' | down e'	(for $\perp$: inject,test,extract)
| λx:t.e' | $e_1(e_2)$ | x	(for $\rightarrow$)
| mkrec e' | unrec e'	(for rec: inject, extract)
| $e_1 \rightarrow e_2, e_3$ | Y e' | f_i	(conditional, fixed points, constants)
| tuple $e_1,\ldots,e_k$ | $take_i$	(for $\underline{\times}$)
| smashtuple $e_1,\ldots,e_k$ | $smashtake_i$	(for $\underline{\otimes}$)
| in_i |case $e_1,\ldots,e_k$	(for $\underline{+}$)

lift e'	up	(for $\underline{\perp}$)
fold	unfold	(for rec)
cond e_1,e_2,e_3	$e_1 \square e_2$	(conditional, composition)

We shall use parentheses freely. The constants f_i are left unspecified but at some points we shall need to assume that certain functions are included. The most common example of this is the identity function id of type $gt \underline{\rightarrow} gt$.

The static well-formedness relation defined below gives the respective types and this may clarify the intention with the constructs. The syntax for top-level constructors is much as in /Sto77/ and /PloLN/. The syntax for bottom-level constructors are in a different style as will be motivated later. It may therefore help to sketch the intended meaning in the standard semantics:

$$(\text{tuple } f_1,f_2)(v_1,v_2) = (f_1(v_1),f_2(v_2))$$

$$(\text{case } f_1,f_2)(\text{in}_i\ v) = f_i(v)$$

$$(\text{lift } f)(\text{up } v) = f(v)$$

$$(\text{cond } f_1,f_2,f_3)(v) = f_1(v) \rightarrow f_2(v), f_3(v)$$

i.e. $\perp$, $f_2(v)$ or $f_3(v)$ depending on $f_1(v) = \perp$, tt or ff

$$(f_1 \square f_2)(v) = f_1(f_2(v))$$

The static well-formedness relation tenv $\vdash$ e : t means that e has type t in type environment tenv. A type environment is a function from a finite subset of identifiers (x, y, x' etc.) to the set of closed top-level types. It will be assumed throughout that t is closed. The relation is defined structurally on e by the following inference rules and axioms (some of which have side conditions).

product $\dfrac{\text{tenv}\vdash e_i:t_i \quad (\text{all } i)}{\text{tenv}\vdash (e_1,\ldots,e_k):t_1\times\ldots\times t_k}$ $\qquad \dfrac{\text{tenv}\vdash e:t_1\times\ldots\times t_k}{\text{tenv}\vdash e\downarrow i:t_i}$

smash product $\dfrac{\text{tenv}\vdash e_i:t_i \quad (\text{all } i)}{\text{tenv}\vdash (*e_1,\ldots,e_k*):t_1*\ldots*t_k}$ $\qquad \dfrac{\text{tenv}\vdash e:t_1*\ldots*t_k}{\text{tenv}\vdash e\downarrow i:t_i}$

sum $\dfrac{\text{tenv}\vdash e:t_i}{\text{tenv}\vdash \text{in}_i\ e:t_1+\ldots+t_k}$ $\qquad \dfrac{\text{tenv}\vdash e:t_1+\ldots+t_k}{\text{tenv}\vdash \text{is}_i\ e:T}$

$\dfrac{\text{tenv}\vdash e:t_1+\ldots+t_k}{\text{tenv}\vdash \text{out}_i\ e:t_i}$

lifting $\dfrac{\text{tenv}\vdash e:t}{\text{tenv}\vdash \text{up}\ e:t_\perp}$ $\qquad \dfrac{\text{tenv}\vdash e:t_\perp}{\text{tenv}\vdash \text{def}\ e:T}$ $\qquad \dfrac{\text{tenv}\vdash e:t_\perp}{\text{tenv}\vdash \text{down}\ e:t}$

function $\dfrac{\text{tenv}[t_1/x]\vdash e:t_2}{\text{tenv}\vdash \lambda x:t_1.e:t_1\rightarrow t_2}$

$\dfrac{\text{tenv}\vdash e_1:t_1\rightarrow t_2 \qquad \text{tenv}\vdash e_2:t_1}{\text{tenv}\vdash e_1(e_2):t_2}$

$\text{tenv}\vdash x:t \quad \text{if } \text{tenv}(x)=t$

recursive types $\dfrac{\text{tenv}\vdash e:t[\text{rec}X.t/X]}{\text{tenv}\vdash \text{mkrec}\ e:\text{rec}X.t}$ $\qquad \dfrac{\text{tenv}\vdash e:\text{rec}X.t}{\text{tenv}\vdash \text{unrec}\ e:t[\text{rec}X.t/X]}$

cond. $\dfrac{\text{tenv}\vdash e_1:T \qquad \text{tenv}\vdash e_2:t \qquad \text{tenv}\vdash e_3:t}{\text{tenv}\vdash e_1\rightarrow e_2,e_3:t}$

fixed $\dfrac{\text{tenv}\vdash e:t\rightarrow t}{\text{tenv}\vdash Y\ e:t}$

const. $\text{tenv}\vdash f_i:t_i \quad \text{if } CP(\emptyset,t_i)$

Let us pause at this point in listing the defining rules for the relation and clarify the notation. The type of truthvalues is T, and $\text{tenv}[t'/x]$ denotes the type environment that is t' upon x and

otherwise as tenv. Syntactic substitution $t[t'/X]$ is defined as in the λ-calculus (see e.g. /Sto77/). To be precise in_i e should be written $in_i(t_1,\ldots,e,\ldots,t_k)$ but the simpler notation should not cause confusion. A similar ambiguity occurs with mkrec because a term $t[recX.t/X]$ needs not determine t uniquely.

The types t_i of constants f_i will not be specified but it is important for the development to restrict the types that are allowed. (An example is the definition of $view_t$ in section 3.3.) This is achieved using the predicate CP(V,t). A type t is said to be <u>contravariantly pure</u> iff CP(Ø,t) holds and the idea is that this is the case iff no ft occurring in t is in the domain of a function space construction. The intention with V is that it contains those domain variables that denote a type that contains some ft. The definition of CP(V,t) is by structural induction on t and uses the auxiliary predicate P(V,t) (for no ft whatsoever).

t	CP(V,t)	P(V,t)
A_i	tt	tt
$t_1 \times \ldots \times t_k$	$\bigwedge_{i=1}^{k} CP(V,t_i)$	$\bigwedge_{i=1}^{k} P(V,t_i)$
...	...	...
$t_1 \rightarrow t_2$	$P(V,t_1) \wedge CP(V,t_2)$	$\bigwedge_{i=1}^{2} P(V,t_i)$
recX.t'	$CP(V \cup \{X\},t') \vee P(V,t')$	P(V,t')
X	tt	$X \notin V$
ft	tt	ff

Some examples may help: $recX.N \rightarrow X \times (\underline{N \rightarrow N})$ is contravariantly pure and so is $recX.N+(X \rightarrow X)$ but $recX.X \rightarrow \underline{N \rightarrow N}$ is not (unfold the definition).

We now continue listing the rules for the well-formedness relation. Recall that $\underline{T}$ is the type of the truth-values in the bottom-level.

$$\text{product}\quad \frac{\text{tenv} \vdash e_i : gt \underline{\rightarrow} gt_i \quad \text{(all i)}}{\text{tenv} \vdash \text{tuple } e_1,\ldots,e_k : gt \underline{\rightarrow} gt_1 \underline{\times} \ldots \underline{\times} gt_k}$$

$$\text{tenv} \vdash \text{take}_i : gt_1 \underline{\times} \ldots \underline{\times} gt_k \underline{\rightarrow} gt_i$$

$$\text{smash}\quad \frac{\text{tenv} \vdash e_i : gt \underline{\rightarrow} gt_i \quad \text{(all i)}}{\text{tenv} \vdash \text{smashtuple } e_1,\ldots,e_k : gt \underline{\rightarrow} gt_1 \underline{*} \ldots \underline{*} gt_k}\quad \text{product}$$

$$\text{tenv} \vdash \text{smashtake}_i : gt_1 \underline{*} \ldots \underline{*} gt_k \underline{\rightarrow} gt_i$$

$$\text{sum}\quad \text{tenv} \vdash \text{in}_i : gt_i \underline{\rightarrow} gt_1 \underline{+} \ldots \underline{+} gt_k$$

$$\frac{\text{tenv} \vdash e_i : gt_i \underline{\rightarrow} gt \quad \text{(all i)}}{\text{tenv} \vdash \text{case } e_1,\ldots,e_k : gt_1 \underline{+} \ldots \underline{+} gt_k \underline{\rightarrow} gt}$$

$$\text{lifting}\quad \frac{\text{tenv} \vdash e : gt \underline{\rightarrow} gt'}{\text{tenv} \vdash \text{lift } e : gt_{\underline{\perp}} \underline{\rightarrow} gt'} \qquad \text{tenv} \vdash \text{up} : gt \underline{\rightarrow} gt_{\underline{\perp}}$$

$$\text{rec.}\quad \text{tenv} \vdash \text{fold} : gt[\underline{\text{rec}}X.gt/X] \underline{\rightarrow} \underline{\text{rec}}X.gt$$

$$\text{tenv} \vdash \text{unfold} : \underline{\text{rec}}X.gt \underline{\rightarrow} gt[\underline{\text{rec}}X.gt/X]$$

$$\text{cond.}\quad \frac{\text{tenv} \vdash e_1 : gt \underline{\rightarrow} \underline{T} \qquad \text{tenv} \vdash e_2 : gt \underline{\rightarrow} gt' \qquad \text{tenv} \vdash e_3 : gt \underline{\rightarrow} gt'}{\text{tenv} \vdash \text{cond } e_1,e_2,e_3 : gt \underline{\rightarrow} gt'}$$

$$\text{comp.}\quad \frac{\text{tenv} \vdash e_1 : gt_2 \underline{\rightarrow} gt_1 \qquad \text{tenv} \vdash e_2 : gt_3 \underline{\rightarrow} gt_2}{\text{tenv} \vdash e_1 \square e_2 : gt_3 \underline{\rightarrow} gt_1}$$

Again there are minor ambiguities. For example in_i should be indexed by $gt_1,\ldots,gt_k$ and perhaps underlined to distinguish it from the similar syntax in the top-level metalanguage.

We now comment upon the syntax of expressions. The syntax relating to the top-level type constructors focuses on elements, as

in $(v_1,...,v_k)$ that is a tuple, whereas the syntax relating to the bottom-level type constructors focuses on functions, as in (tuple $f_1,...,f_k$) that is a function. It is possible to use a syntax focusing on functions throughout (see /PloLN/) but it seems to be more common to focus on elements (e.g. /Sto77/). For the syntax relating to bottom-level types it seems necessary to focus on functions, e.g. in the definition of cond in section 3.3. This is in accord with abstract interpretation where it is certain "state transformation" functions that are of interest. Also note that the constructs tuple, smashtuple, case, lift, cond and □ may be viewed as <u>functionals</u> of type $ft^n \rightarrow ft$. They are the prime examples of "constants" whose types are not contravariantly pure. In particular there are no constructs of type $ft \rightarrow T$ (or $gt \rightarrow T$) so in a sense the dynamic semantics cannot influence the static semantics.

2.2 PRELIMINARY DOMAIN THEORY

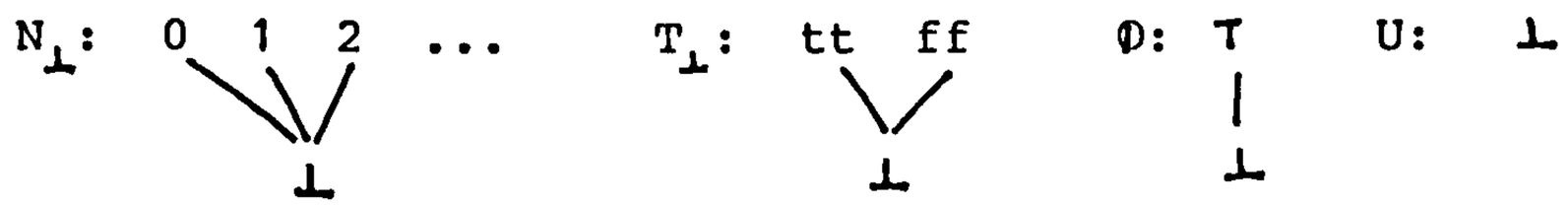
This section reviews/develops the domain theory needed to define the standard semantics of the metalanguage. Most of the material is quite standard and can be found elsewhere, e.g. in /SmP182/ in a terse presentation and in /PloLN/. The material originates with Dana Scott who gives a "modern" presentation of the theory in /Sco82/ but in a different flavour than used here. Less standard results will be proved but much will be stated without proof. A few concepts from category theory /ArMa75/ are used but no previous knowledge is assumed.

BASIC ORDER THEORETIC DEFINITIONS

Partially ordered sets, complete lattices and chains were defined in chapter 1. A cpo (countably complete partial order) is a partially ordered set $D = (D, \sqsubseteq)$ with a least element and least uppper bounds of chains. A subset H of D is directed if each finite subset of H has an upper bound in H. Then one can prove that a partially ordered set is a cpo iff it has a least element and every countable directed set has a least upper bound. A cpo D is flat iff the formula

$$d \sqsubseteq d' \Leftrightarrow (d = \bot \vee d = d')$$

is true. Example flat cpo's are:



In general, for any set S there is the flat cpo $S_\bot$ with elements $S \cup \{\bot\}$ (assuming a disjoint union). The set $\emptyset_\bot$ will be of special interest and is denoted U. We shall sometimes write $\mathcal{O}$ for $\{T\}_\bot$.

An element d of a cpo D is finite iff for every chain $(d_n)_n$ we have that $d \sqsubseteq \bigsqcup_n d_n$ implies that $d \sqsubseteq d_n$ for some n. The idea is that finite elements correspond to approximative information about fully defined elements /Sco82/. This is exemplified by the cpo of partial functions from N to N ordered by subset on the graphs of the functions. Here the finite elements are the partial functions that have a finite domain. For this cpo every element is the least upper bound of a chain of finite elements. This is often the case and motivates:

Definition A cpo D is countably algebraic (or just algebraic) if the set $B_D = \{b \in D \mid b \text{ is finite}\}$ is countable and each $d \in D$ has

$d = \bigsqcup_n b_n$ for some chain $(b_n)_n$ with each $b_n \in B_D$. ///

We shall mostly assume cpo's to be algebraic as this simplifies some constructions; this assumption is made in many developments of domain theory. One can show that in an algebraic cpo every directed set has a least upper bound. Clearly $S_\perp$ is algebraic iff S is countable. It is convenient to know:

Fact 2.2:1. A cpo D is algebraic iff there is a countable subset H whose elements are finite and where each element d of D can be written $\bigsqcup H'$ for a directed subset H' of H. The set H then equals B_D. ///

Not all constructions upon cpo's that we will consider preserve algebraicity. We therefore define an additional property and we shall see that the conjunction of the two properties is preserved.

Definition A cpo D is consistently complete iff $\bigsqcup H$ exists for every subset H of D that has an upper bound in D. ///

A cpo D is consistently complete iff the partially ordered set D' obtained by adding a new greatest element to D is a complete lattice. It is generally believed that in practice cpo's are algebraic and consistently complete; such cpo's are often called domains (essentially as in /Sco82/). On the other hand it is often unnatural to assume cpo's to be complete lattices because the "artificial" greatest element may invalidate equalities that hold operationally (see e.g. /PloLN/).

We now define several ways of defining a cpo in terms of others. Let $D, D_1, \ldots, D_k$ be cpo's and assume that $k \geq 2$. The cartesian

<u>product</u> $D_1 \times \ldots \times D_k$ or $\prod_i D_i$ has as elements tuples $(d_1,\ldots,d_k)$ with $d_i \in D_i$ and is partially ordered componentwise, i.e.

$$(d_1,\ldots,d_k) \sqsubseteq (d_1',\ldots,d_k') \text{ iff for all i that } d_i \sqsubseteq d_i'$$

This gives a cpo with $\sqcup$ and $\sqcap$ defined componentwise and it is algebraic/consistently-complete/a domain if all D_i are. We write $(d_1,\ldots,d_k)\downarrow i$ for d_i. Closely associated with the cartesian product is the <u>smash product</u> $D_1 * \ldots * D_k$. The elements are those tuples $(d_1,\ldots,d_k)$ where some d_i is $\perp$ iff all are. The partial order is as before and defines a cpo that is algebraic / consistently complete/ a domain if all D_i are. The function $\text{smash}: D_1 \times \ldots \times D_k \to D_1 * \ldots * D_k$ sends $(d_1,\ldots,\perp,\ldots,d_k)$ to $(\perp,\ldots,\perp)$ and otherwise acts as the identity.

The <u>coalesced sum</u> $D_1 + \ldots + D_k$ or $\sum_i D_i$ has as elements pairs (i,d_i) with $d_i \in D_i$ and $d_i \neq \perp$ and additionally the element $\perp$. The partial order has $\perp$ to be least and $(i,d_i) \sqsubseteq (j,d_j)$ iff $i=j$ and $d_i \sqsubseteq d_j$. This gives a cpo that is algebraic/ consistently complete/ a domain if all D_i are. The function $\text{in}_i: D_i \to D_1 + \ldots + D_k$ sends $\perp$ to $\perp$ and otherwise d_i to (i,d_i). Similarly $\text{out}_i: D_1 + \ldots + D_k \to D_i$ sends (i,d_i) to d_i and otherwise gives $\perp$. Finally, $\text{is}_i: D_1 + \ldots + D_k \to T$ sends (i,d_i) to tt and $\perp$ to $\perp$ and otherwise gives ff.

The <u>lifting</u> $D_\perp$ has as elements a new least element $\perp$ and for each element $d \in D$ the element $(0,d)$. The partial order has $(0,d) \sqsubseteq (0,d')$ iff $d \sqsubseteq d'$. This gives a cpo that is algebraic/ consistenly complete/ a domain if D is. The function $\text{up}: D \to D_\perp$ sends d to $(0,d)$ and $\text{down}: D_\perp \to D$ sends $\perp$ to $\perp$ and $(0,d)$ to d. Finally, $\text{def}: D_\perp \to T$ sends $\perp$ to $\perp$ and $(0,d)$ to tt.

Before defining function spaces we explain the notion of continuity. A function $f:D_1 \to D_2$ is <u>continuous</u> iff it is monotonic and for each chain $(d_n)_n$ of D_1 it is the case that $f(\bigsqcup_n d_n) = \bigsqcup_n f(d_n)$. Note that the monotonicity of f ensures that $(f(d_n))_n$ is a chain in D_2 and that $f(\bigsqcup_n d_n) \sqsupseteq \bigsqcup_n f(d_n)$. It is helpful to know that $f(\bigsqcup H) = \bigsqcup\{f(h) \mid h \in H\}$ holds when f is continuous and H is countable and directed. If D_1 is algebraic this holds even when H is not countable. It is not hard to show that $f:D_1 \times \ldots \times D_k \to D$ is continuous iff for all $d_1, \ldots, d_k$ and i the equation $g(d) = f(d_1,\ldots,d,\ldots,d_k)$ defines a continuous function $g:D_i \to D$. All the functions defined so far in this section are continuous and it is generally believed that all functions should be continuous /Sco82/.

The <u>function space</u> $D_1 \to D_2$ has as elements all continuous functions from D_1 to D_2. The partial order is defined pointwise, i.e. $f \sqsubseteq f'$ iff $f(d) \sqsubseteq f'(d)$ holds for all elements d of D_1. This gives a cpo with least upper bounds determined pointwise. It is a domain if D_1 and D_2 are but needs not be algebraic if D_1 and D_2 are only algebraic /PloLN/.

If f is a continuous function upon a cpo it has a least fixed point $LFP(f) = \bigsqcup_n f^n(\bot)$. Here f^n is the n-fold composition of f and the monotonicity of f gives that $(f^n(\bot))_n$ is a chain. Clearly $\bigsqcup_n f^n(\bot)$ is a fixed point and to see it is less than any other fixed point d one shows $f^n(\bot) \sqsubseteq d$ by induction on n and then deduces that $\bigsqcup_n f^n(\bot) \sqsubseteq d$. The formula $LFP(f) = \bigsqcap\{d \mid f(d) \sqsubseteq d\}$ suggested by /Tar55/ still holds: For $\bigsqcup_n f^n(\bot)$ is an element of the set on the righthand side and any other element d has $\bigsqcup_n f^n(\bot) \sqsubseteq d$ as can be shown by a proof similar to the proof that $\bigsqcup_n f^n(\bot)$ is the least fixed point.

Using $LFP(f) = \bigsqcup_n f^n(\bot)$ it is not hard to show that LFP is a continuous function from $D \to D$ to D (see e.g. /Sto77/).

The proof that $\bigsqcup_n f^n(\bot)$ is the least fixed point is typical of many proofs. To facilitate such proofs one can use the following proof rule. A predicate P on the cpo D is <u>admissible</u> (inclusive) iff $P(\bot)$ is true and if for every chain $(d_n)_n$ the truth of all $P(d_n)$ gives the truth of $P(\bigsqcup_n d_n)$. Then for a continuous $f: D \to D$ and admissible P we have the rule of <u>fixed point induction</u> (Scott-induction)

$$\frac{P(x) \Rightarrow P(f(x))}{P(LFP(f))}$$

For since P is admissible $P(\bot)$ holds and induction on n shows that all the $P(f^n(\bot))$ hold and hence $P(\bigsqcup_n f^n(\bot))$ holds. (Often $P(\bot)$ is made a premise of the rule and then not assumed in the definition of admissibility.) As an example the predicate P(x) defined by $\forall d: (f(d) \sqsubseteq d \Rightarrow x \sqsubseteq d)$ may be used to prove that $LFP(f) \sqsubseteq d$ whenever $f(d) \sqsubseteq d$.

CATEGORICAL FORMULATION

The remainder of this section is devoted to the solution of (parameterised) recursive domain equations. The development is closer to /SmP182/ and /PloLN/ than to the universal domain approach /Sco76/. This calls for the use of some categorical concepts to expedite the development. The necessary concepts are explained below and the constructs defined earlier are formulated in this setting. The two central concepts are those of category and functor /ArMa75,Mac71/.

A category allows for a concise way of naming the cpo's and functions of interest at some point in the development. A category consists of a set of objects and for each two objects A and B a set of morphisms $g:A\rightarrow B$ from A to B. (Some authors require that a morphism like g must uniquely determine its domain A and range B.) The most common category is $\underline{\underline{\underline{Set}}}$ that has sets as objects and total functions as morphisms. (Strictly speaking not all sets can be allowed as the class of objects then is "too big" to be a set; ways around this are discussed in /Mac71 p.21-24/.) We shall make much use of $\underline{\underline{\underline{CPO}}}$ that has cpo's as objects and (total) continuous functions as morphisms. To be a category there must be a composition of morphisms, as in $f\cdot g:A\rightarrow C$ that is the composition of $f:B\rightarrow C$ and $g:A\rightarrow B$, and for each object A an identity morphism $id_A:A\rightarrow A$ (usually written id). They must satisfy the associative law $(f\cdot g)\cdot h=f\cdot(g\cdot h)$ and the identity laws $f\cdot id=f$ and $id\cdot f=f$. In $\underline{\underline{\underline{Set}}}$ and $\underline{\underline{\underline{CPO}}}$ composition is ordinary function composition and the identities are the identity functions. For $\underline{\underline{\underline{CPO}}}$ one must check that the identity function is continuous and that composition of continuous functions gives a continuous function.

It is convenient to name some additional categories. If only strict continuous functions are allowed the category is $\underline{\underline{\underline{\underline{CPOs}}}}$. If all continuous functions are allowed but object must be domains (i.e. algebraic and consistently complete cpo's) the category is $\underline{\underline{\underline{ACC}}}$. Similarly $\underline{\underline{\underline{\underline{ACCs}}}}$ requires functions to be strict.

The category $\underline{\underline{\underline{\underline{ACCs}}}}$ is said to be a subcategory of $\underline{\underline{\underline{CPO}}}$ because all objects and morphisms of the former are in the latter and because the composition and identities agree. It is also a subcategory of

$\underline{\underline{\underline{ACC}}}$ and $\underline{\underline{\underline{CPOs}}}$. A product category like $\underline{\underline{\underline{CPO}}} \times \underline{\underline{\underline{CPO}}}$ or $\underline{\underline{\underline{CPO}}}^k$ has objects and morphisms to be tuples with one component for each category in the product. In particular $\underline{\underline{\underline{CPO}}}^1$ may be identified with $\underline{\underline{\underline{CPO}}}$ and $\underline{\underline{\underline{CPO}}}^0$ is a category with one object NIL and one morphism id_{NIL}. If $f_1:A_1 \rightarrow B_1$ and $f_2:A_2 \rightarrow B_2$ are morphisms then $(f_1,f_2):(A_1,A_2) \rightarrow (B_1,B_2)$ is a morphism in the product category. Composition and identities are defined componentwise, i.e. $(f_1,f_2)\cdot(g_1,g_2) = (f_1 \cdot g_1, f_2 \cdot g_2)$ and $id_{(A_1,A_2)} = (id_{A_1}, id_{A_2})$.

A covariant functor (sometimes abbreviated to functor) allows for a precise description of the effect of domain constructors (e.g. lifting). A covariant functor F from a category $\underline{\underline{\underline{A}}}$ to a category $\underline{\underline{\underline{B}}}$ consists of two maps: one that sends an object A of $\underline{\underline{\underline{A}}}$ to an object F(A) of $\underline{\underline{\underline{B}}}$ and one that sends a morphism $f:A_1 \rightarrow A_2$ of $\underline{\underline{\underline{A}}}$ to a morphism $F(f):F(A_1) \rightarrow F(A_2)$ of $\underline{\underline{\underline{B}}}$. It must satisfy the composition law $F(f \cdot g) = F(f) \cdot F(g)$ and the identity law $F(id_A) = id_{F(A)}$.

It is now straightforward to turn lifting, cartesian product, smash product and coalesced sum into covariant functors. For lifting we define a functor $(\ )_\perp:\underline{\underline{\underline{CPO}}} \rightarrow \underline{\underline{\underline{CPO}}}$ as follows. The effect upon objects has already been defined and for a morphism $f:D \rightarrow E$ the functor gives $(f)_\perp:D_\perp \rightarrow E_\perp$ that sends $\perp$ to $\perp$ and (0,d) to (0,f(d)). It is straightforward to check that the composition and identity laws hold. It will become clear when solving domain equations that it is helpful that the domain constructors are defined as functors, i.e. have an effect upon morphisms.

Cartesian product is turned into a covariant functor $\times:\underline{\underline{\underline{CPO}}}^k \rightarrow \underline{\underline{\underline{CPO}}}$ by defining $f_1 \times \ldots \times f_k$ as the function that maps $(v_1,\ldots,v_k)$ to $(f_1(v_1),\ldots,f_k(v_k))$. These functors preserve strictness and therefore

specialise to $\underline{\underline{\text{CPOs}}}$ (i.e. we could have used $\underline{\underline{\text{CPOs}}}$ above). To turn smash product and coalesced sum into covariant functors we need to assume strictness, i.e. they become functors $\underline{\underline{\text{CPOs}}}^k \to \underline{\underline{\text{CPOs}}}$. Smash product has $f_1 * \ldots * f_k$ to be the restriction of smash$\cdot f_1 \times \ldots \times f_k$ to the smash product. Coalesced sum has $f_1 + \ldots + f_k$ to be the function mapping (i,d_i) to $(i,f_i(d_i))$ when $f_i(d_i) \neq \perp$ and giving $\perp$ in all other cases. All of these functors preserve algebraicity and consistent completeness and therefore specialise to $\underline{\underline{\text{ACCs}}}$.

We shall consider two ways of combining functors. The tupling of covariant functors $F_i : \underline{\underline{L}} \to \underline{\underline{K}}_i$ is the covariant functor $(F_1,\ldots,F_k) : \underline{\underline{L}} \to \underline{\underline{K}}_1 \times \ldots \times \underline{\underline{K}}_k$ defined by $(F_1,\ldots,F_k)(A) = (F_1(A),\ldots,F_k(A))$ and similarly on morphisms. (So $(F_1,\ldots,F_k)$ is a functor and not a tuple of functors!) The composition of covariant functors $F_1 : \underline{\underline{L}} \to \underline{\underline{K}}$ and $F_2 : \underline{\underline{K}} \to \underline{\underline{M}}$ is the covariant functor $F_2 \cdot F_1 : \underline{\underline{L}} \to \underline{\underline{M}}$ defined on objects by $(F_2 \cdot F_1)(A) = F_2(F_1(A))$ and similarly on morphisms. The covariant identity functor Id maps an object to itself and similarly for morphisms. If A is an object of $\underline{\underline{L}}$ then a covariant constant functor $K_A : \underline{\underline{M}} \to \underline{\underline{L}}$ is defined by $K_A(B) = A$ and $K_A(f) = id_A$. The i'th projection functor $P_i : \underline{\underline{K}}_1 \times \ldots \times \underline{\underline{K}}_k \to \underline{\underline{K}}_i$ is the covariant functor defined by $P_i(A_1,\ldots,A_k) = A_i$ and similarly on morphisms.

The function space construction is more troublesome. The effect upon objects has already been defined and upon morphisms one may define $f_1 \to f_2$ as the function sending h to $f_2 \cdot h \cdot f_1$. The identity law holds but for composition we get

$$(f_1 \to f_2) \cdot (g_1 \to g_2) = g_1 \cdot f_1 \to f_2 \cdot g_2$$

corresponding to $(f_1 \to f_2) : (B_1 \to A_2) \to (A_1 \to B_2)$ when $f_i : A_i \to B_i$. This functor is said to be contravariant in its left argument and

covariant in its right. It gives a functor $\rightarrow:\underline{\underline{CPO}}^2\rightarrow\underline{\underline{CPO}}$ that specialises to $\underline{\underline{CPOs}}$, $\underline{\underline{ACC}}$ and $\underline{\underline{ACCs}}$.

Contravariance is troublesome to work with, e.g. because $\rightarrow^{\bullet}(Id,Id)$ is neither contravariant nor covariant. To enable all functors to be covariant we shall use the category $\underline{\underline{CPO2s}}$ instead of $\underline{\underline{CPOs}}$. This category has the same objects as $\underline{\underline{CPOs}}$ but a morphism $f:A\rightarrow B$ is a pair (f_1,f_2) of $\underline{\underline{CPOs}}$ morphisms $f_1:A\rightarrow B$ and $f_2:B\rightarrow A$. In other words a morphism of $\underline{\underline{CPO2s}}$ is a pair of strict continuous functions. Composition is defined by

$$(f_1,f_2)^{\bullet}(g_1,g_2) = (f_1{}^{\bullet}g_1,g_2{}^{\bullet}f_2)$$

and may be viewed as being contravariant in its right argument. It is this property that will allow contravariance over $\underline{\underline{CPOs}}$ to be disguised as covariance over $\underline{\underline{CPO2s}}$. The identity morphism on A is the pair (id,id) of identity functions $id:A\rightarrow A$. When $f = (f_1,f_2)$ it is convenient to write $f\downarrow i = f_i$ and $f^R = (f_2,f_1)$.

A <u>symmetric functor</u> /PloPS/ (or a <u>domain functor</u> /Rey74/) $G:\underline{\underline{CPO2s}}^k\rightarrow\underline{\underline{CPO2s}}$ is a covariant functor that satisfies the law

$$G(f_1,\ldots,f_k)^R = G(f_1{}^R,\ldots,f_k{}^R)$$

When k=1 this means that $G((g_1,g_2)) = (h_1,h_2)$ implies that $G((g_2,g_1)) = (h_2,h_1)$. Let $F:\underline{\underline{CPOs}}^k\rightarrow\underline{\underline{CPOs}}$ be a mixed covariant and contravariant functor. A symmetric functor $F^S:\underline{\underline{CPO2s}}^k\rightarrow\underline{\underline{CPO2s}}$ may be defined by

$$F^S(A_1,\ldots,A_k) = F(A_1,\ldots,A_k)$$
$$F^S(f_1,\ldots,f_k) = (F(f_1\downarrow j_1,\ldots,f_k\downarrow j_k), F(f_1\downarrow(3-j_1),\ldots,f_k\downarrow(3-j_k)))$$

where $j_i=1$ if F is covariant in the i'th argument and otherwise $j_i=2$. This means that all of $\times$, $*$, $+$, $(\ )_\perp$, $\rightarrow$ may be viewed as symmetric

functors over $\underline{\underline{\underline{\underline{\underline{CPO2s}}}}}$. For example

$$+^S((f_1,f_2) , (g_1,g_2)) = (f_1+g_1 , f_2+g_2)$$

$$\rightarrow^S((f_1,f_2) , (g_1,g_2)) = (f_2\rightarrow g_1 , f_1\rightarrow g_2)$$

The use of $\underline{\underline{\underline{\underline{\underline{CPO2s}}}}}$ rather than $\underline{\underline{\underline{\underline{CPO2}}}}$ is because + and $\ast$ were only defined as functors over $\underline{\underline{\underline{\underline{CPOs}}}}$. Also the identity functor, constant functors and projection functors can be transformed to $\underline{\underline{\underline{\underline{\underline{CPO2s}}}}}$ using ^{S} and they "give themselves".

Most of the categories considered so far have some additional structure. A cpo-category is a category where the set of morphisms between any two objects forms a cpo and where composition is continuous with respect to the partial orders. (This corresponds to a $\underline{\underline{\underline{CPO}}}$-category in /ArMa75/.) For $\underline{\underline{\underline{\underline{CPOs}}}}$ the partial order is defined pointwise (as in the function space construct), for $\underline{\underline{\underline{\underline{\underline{CPO2s}}}}}$ it is defined componentwise (i.e. $(f_1,f_2)\sqsubseteq(g_1,g_2)$ iff $f_1\sqsubseteq g_1$ and $f_2\sqsubseteq g_2$) and for products of categories is defined componentwise (as in the cartesian product of cpo's). A functor between two cpo-categories is locally continuous / locally monotonic if its effect upon morphisms is continuous / monotonic. For a (mixed covariant and contravariant) functor $F:\underline{\underline{\underline{\underline{CPOs}}}}^k\rightarrow\underline{\underline{\underline{\underline{CPOs}}}}$ local continuity just means that

$$F(\bigsqcup_n f_n, \ldots, \bigsqcup_n h_n) = \bigsqcup_n F(f_n,\ldots,h_n)$$

for all chains $(f_n)_n$, ..., $(h_n)_n$ of strict continuous functions. Clearly local continuity implies local monotonicity and if $F:\underline{\underline{\underline{\underline{CPOs}}}}^k\rightarrow\underline{\underline{\underline{\underline{CPOs}}}}$ is locally continuous / locally monotonic then so is $F^S:\underline{\underline{\underline{\underline{\underline{CPO2s}}}}}^k\rightarrow\underline{\underline{\underline{\underline{\underline{CPO2s}}}}}$. All the functors $(\ )_\perp$, $\times$, $\ast$, $+$, $\rightarrow$, Id, K_A, P_i are locally continuous and composition and tupling preserve this property.

RECURSIVE DOMAIN EQUATIONS

The notion of an isomorphism is essential to explain what is meant by a solution to a recursive domain equation. An <u>isomorphism</u> in a category is a morphism $\theta:A\rightarrow B$ for which there exists a (necessarily unique) morphism $\theta^{-1}:B\rightarrow A$ such that $\theta\cdot\theta^{-1}=id_A$ and $\theta^{-1}\cdot\theta=id_B$. The morphism θ^{-1} is called the inverse of θ and is itself an isomorphism with inverse $(\theta^{-1})^{-1}=\theta$. We write $A\cong B$ if there is an isomorphism from A to B. The identity is clearly an isomorphism with itself as inverse and the composition of isomorphisms give an isomorphism and one has the formula $(\theta_1\cdot\theta_2)^{-1} = \theta_2^{-1}\cdot\theta_1^{-1}$. If F is a covariant functor and $\theta_1, \ldots, \theta_k$ are all isomorphisms then so is $F(\theta_1,\ldots,\theta_k)$ and its inverse is $F(\theta_1^{-1},\ldots,\theta_k^{-1})$. The isomorphisms in $\underline{\underline{\underline{CPO}}}$, $\underline{\underline{\underline{\underline{CPOs}}}}$, $\underline{\underline{\underline{ACC}}}$, $\underline{\underline{\underline{\underline{ACCs}}}}$ are the bijections such that it and its inverse are monotonic. In $\underline{\underline{\underline{\underline{\underline{CPO2s}}}}}$ and $\underline{\underline{\underline{\underline{CPO2}}}}$ isomorphisms are pairs of such functions.

For recursive domain equations we consider only systems of one equation. Systems of several equations can be solved too but are not needed for the semantics of the metalanguage. (The effect of several equations can be achieved by solving several parameterised equations /LeSm81/ so little generality is lost.) A domain equation like

$$X = T + (X \times N)$$

corresponding to

$$\text{rec } X.\ T + (X \times N)$$

in the metalanguage gives rise to a covariant functor

$$+ \cdot (K_T\ ,\ \times \cdot (Id\ ,\ K_N))$$

over $\underline{\underline{\underline{\underline{CPOs}}}}$. To cater for occurrences of function space we shall use

$$+^S \cdot (K_T , \times^S \cdot (Id , K_N))$$

over $\underline{\underline{\underline{\underline{\underline{CPO2s}}}}}$ instead. The analogue of a fixed point of a function is captured by:

Definition /SmPl82,PloLN/. The pair (X,Θ) is a fixed point of a covariant functor G over some category iff X is an object and $\Theta:G(X)\rightarrow X$ is an isomorphism. ///

If F is a covariant functor over $\underline{\underline{\underline{\underline{CPOs}}}}$ the pair (X,Θ) is a fixed point of F iff $(X,(\Theta,\Theta^{-1}))$ is a fixed point of F^S. Equality, i.e. $\Theta=id_X$, is usually not considered because the axiom of regularity of ZF set theory shows that $X = X \times X$ cannot hold when $X \neq \emptyset$ /Ham82/.

The notion of an embedding is essential to explain the analogue of least fixed points. An embedding in a cpo-category $\underline{\underline{B}}$ is a morphism $e:A\rightarrow B$ for which there exists another morphism $e^U:B\rightarrow A$ such that $e^U \cdot e = id_A$ and $e \cdot e^U \sqsubseteq id_B$. The morphism e^U is the upper adjoint of e and is unique if it exists. Every isomorphism Θ is an embedding and $\Theta^U = \Theta^{-1}$. The composition of embeddings gives an embedding and $(e_1 \cdot e_2)^U = e_2^U \cdot e_1^U$. Therefore one obtains a subcategory $\underline{\underline{Be}}$ of $\underline{\underline{B}}$ by restricting the morphisms to be the embeddings of $\underline{\underline{B}}$. The categories $\underline{\underline{\underline{\underline{CPOe}}}}$ and $\underline{\underline{\underline{\underline{\underline{CPOse}}}}}$ are the same because an embedding of $\underline{\underline{\underline{CPO}}}$ is completely additive (hence strict) with a strict upper adjoint. An embedding in $\underline{\underline{\underline{\underline{\underline{CPO2s}}}}}$ is a pair (e_1,e_2^U) where e_1 and e_2 are embeddings of $\underline{\underline{\underline{\underline{CPOs}}}}$ and the upper adjoint is (e_1^U,e_2). A locally monotonic covariant functor specialises to a functor upon the subcategories of embeddings because if $e_1, \ldots, e_k$ are embeddings then so is $F(e_1,\ldots,e_k)$ with upper adjoint $F(e_1^U,\ldots,e_k^U)$. Finally, the formula $e_1 \sqsubseteq e_2$ iff $e_1^U \sqsupseteq e_2^U$

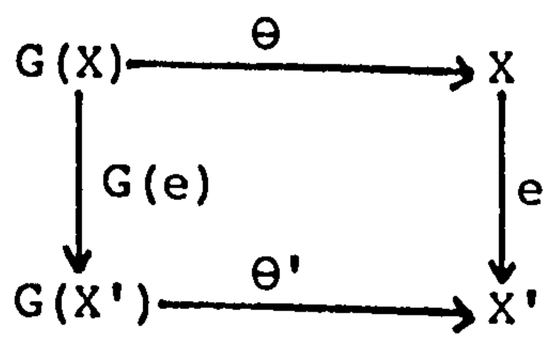
holds for embeddings e_1 and e_2.

The existence of an embedding $e:A \to B$ may be viewed as an analogue of $a \sqsubseteq b$ for elements of some partially ordered set. Consult /PloLN/ for this kind of motivation. Using the categorical concept initiality the analogue of a <u>least fixed point</u> is captured by the following definition that is explained afterwards.

<u>Definition</u> The pair (X,Θ) is an <u>initial fixed point</u> of a covariant functor G over some cpo-category iff it is a fixed point and for every fixed point (X',Θ') there exists precisely one embedding $e:X \to X'$ such that $e \cdot \Theta = \Theta' \cdot G(e)$. ///

If F is a covariant functor over CPOs the pair (X,Θ) is an initial fixed point of F iff $(X,(\Theta,\Theta^{-1}))$ is an initial fixed point of F^S.

The equation $e \cdot \Theta = \Theta' \cdot G(e)$ may be formulated as the requirement that the diagram



commutes. Intuitively, the diagram requires the embedding e of interest to identify elements of X with elements of X' in a way consistent with how G was built. (It is essential for the diagram to make sense that G is covariant.) The existence of precisely one embedding means that X has the "right" collection of elements. Even when initial fixed points exist they need not be unique. However, any two initial fixed points (X,Θ) and (X',Θ') are isomorphic in the sense that there is an isomorphism $\varphi:X \to X'$ such that $\varphi \cdot \Theta = \Theta' \cdot G(\varphi)$. For the proof construct φ as the unique embedding from X to X' as

detailed in the definition. Similarly φ^{-1} is constructed from X' to X. That $\varphi^{-1}\cdot\varphi = id_X$ follows because both $\varphi^{-1}\cdot\varphi$ and id_X are embeddings that can be used from X to X as detailed in the definition. Similarly $\varphi\cdot\varphi^{-1} = id_X$, and it follows that φ is an isomorphism.

Solving in CPOe

Let G be a locally monotonic symmetric functor over CPO2s corresponding to some domain equation that is to be solved. We have already seen that G specialises to a covariant functor over CPO2e and a covariant functor G^E:CPOe→CPOe is obtained by

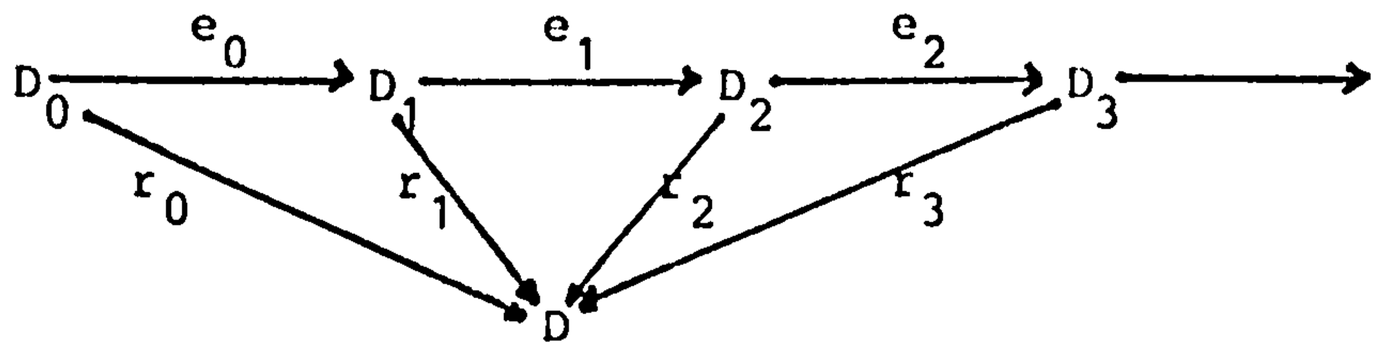
$$G^E(A) = G(A)$$

$$G^E(e) = G((e,e^U))\downarrow 1$$

This setting also includes a locally monotonic functor F over CPOs as then $G = F^S$ and G^E is the specialisation of F to CPOe. It is convenient to begin with considering an initial fixed point for G^E. Later it will be transformed to CPO2s and certain sub-categories of CPOs.

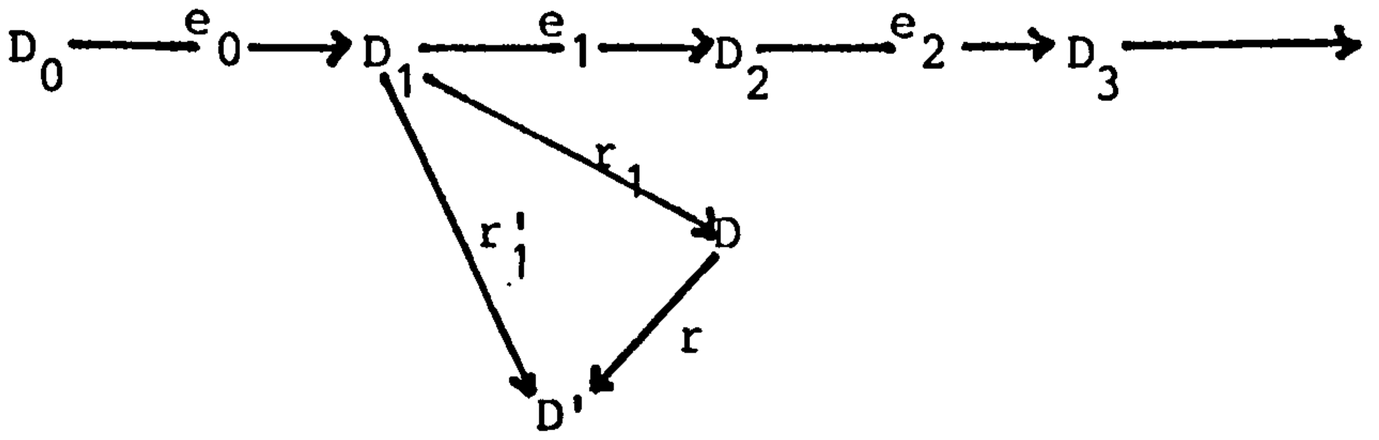
Corresponding to a chain of elements of a cpo one may define a notion of a chain in a category. It is a pair $\underline{E} = ((D_n)_n,(e_n)_n)$ where each D_n is an object and each $e_n:D_n \to D_{n+1}$ is a morphism. A chain in CPOe therefore is a pair $((D_n)_n,(e_n)_n)$ where each D_n is a cpo and each e_n is an embedding of CPOs. The analogue of an upper bound for the chain $\underline{E}$ is that of a cone /PloLN/. (In strict categorical language the prefix co- should be used.) A cone is a pair $\underline{R} = (D,(r_n))$ where D is an object and each $r_n:D_n \to D$ is a morphism such that $r_{n+1}\cdot e_n = r_n$. This may be illustrated by the

diagram



and the requirement that all small triangles (hence all triangles) commute.

The analogue of a least upper bound is called a limiting cone /PloLN/ and builds on the categorical idea of initiality. The cone $\underline{R}$ is <u>limiting</u> (or initial) iff for every cone $\underline{R}' = (D', (r'_n)_n)$ there is precisely one morphism $r:D \rightarrow D'$ such that $r \cdot r_m = r'_m$. The morphism r is said to <u>mediate</u> from $\underline{R}$ to $\underline{R}'$ /SmPl82/. This may be illustrated by the diagram



and the requirement that all such triangles must commute. If a chain has a limiting cone it may well have more than one. If $\underline{R}$ and $\underline{R}'$ are limiting cones they are isomorphic in the sense that there is an isomorhism $\theta:D \rightarrow D'$ such that $\theta \cdot r_n = r'_n$. To see this construct θ as the mediating morphism from $\underline{R}$ to $\underline{R}'$ and θ^{-1} as the mediating morphism from $\underline{R}'$ to $\underline{R}$. Since both $\theta^{-1} \cdot \theta$ and id_D mediate from $\underline{R}$ to $\underline{R}$ it follows that $\theta^{-1} \cdot \theta = id_D$ and $\theta \cdot \theta^{-1} = id_{D'}$ is shown similarly.

In $\underline{\underline{\underline{\underline{\underline{CPOe}}}}}$ a chain always has a limiting cone and it is easy to give a non-categorical criterion for when a cone is limiting.

Theorem 2.2:2 /PloLN/ (essentially /SmP182/).

Let $\underline{E} = ((D_n)_n, (e_n)_n)$ be a chain in $\underline{\underline{CPOe}}$ and $\underline{R} = (D, (r_n)_n)$ a cone for $\underline{E}$. Then

(1) $\underline{R}$ is a limiting cone iff $id_D = \bigsqcup_n r_n \cdot r_n^U$

(2) $\underline{E}$ always has a limiting cone

(3) If $\underline{R}$ is limiting and $\underline{R}'$ is a cone then the mediating morphism from $\underline{R}$ to $\underline{R}'$ is $r = \bigsqcup_n r'_n \cdot r_n^U$ with upper adjoint $\bigsqcup_n r_n \cdot r'^U_n$. ///

...

Sketch of proof.. First note that for any two cones $\underline{R}$ and $\underline{R}'$ the sequence $(r'_n \cdot r_n^U)_n$ is a chain and therefore $\bigsqcup_n r'_n \cdot r_n^U$ is well-defined as a function. Next let $\underline{R}$ be a cone such that $id_D = \bigsqcup_n r_n \cdot r_n^U$. Then $r = \bigsqcup_n r'_n \cdot r_n^U$ is an embedding with $\bigsqcup_n r_n \cdot r'^U_n$ as its upper adjoint. It is immediate that r mediates from $\underline{R}$ to $\underline{R}'$. If also r' mediates from $\underline{R}$ to $\underline{R}'$ we have

$$r' = r' \cdot id_D = \bigsqcup_n r' \cdot r_n \cdot r_n^U = \bigsqcup_n r'_n \cdot r_n^U = r$$

This shows "if" in (1) and (3) in a special case.

To show (2) we construct a cone $\underline{R}$ such that $id_D = \bigsqcup_n r_n \cdot r_n^U$.

Let

$$D = \{(d_0, d_1, \ldots) \mid d_n \in D_n \wedge d_n = e_n^U(d_{n+1})\}$$

When partially ordered componentwise this gives a cpo. Further let

$$r_n(d_n) = (\ldots, e_{n-1}^U(d_n), d_n, e_n(d_n), \ldots)$$

and note that $r_n^U((d_0, d_1, \ldots)) = d_n$. Since $\bigsqcup_n r_n \cdot r_n^U = id$ it follows that $\underline{R}$ is limiting. It remains to show "only if" in (1) and (3) in the general case. This may be shown using the "isomorphism" between a given limiting cone $\underline{R}'$ and the $\underline{R}$ constructed above. ///

It is convenient to define the effect of G^E upon chains and cones. For a chain $\underline{E} = ((D_n)_n, (e_n)_n)$ a chain $G^E[\![\underline{E}]\!]$ is defined by

$$G^E[\![\underline{E}]\!] = ((G^E(D_n))_n, (G^E(e_n))_n)$$

and for a cone $\underline{R} = (D, (r_n)_n)$ a cone $G^E[\![\underline{R}]\!]$ is defined by

$$G^E[\![\underline{R}]\!] = (G^E(D), (G^E(r_n))_n)$$

and it is a cone for $G^E[\![\underline{E}]\!]$. To preserve limits we shall assume G to be locally continuous:

Lemma 2.2:3 /PloLN/ (essentially /SmPl82/).

Let G be a locally continuous symmetric functor and $\underline{E}$ a chain in $\underline{\underline{CPOe}}$ with limiting cone $\underline{R} = (D, (r_n)_n)$. Then $G^E[\![\underline{R}]\!]$ is a limiting cone for $G^E[\![\underline{E}]\!]$ and if r mediates from $\underline{R}$ to a cone $\underline{R}' = (D', (r'_n)_n)$ then the mediating morphism from $G^E[\![\underline{R}]\!]$ to $G^E[\![\underline{R}']\!]$ is $G^E(r)$. ///

This result is a straightforward consequence of the previous theorem and may be formulated as

$$G^E(\bigsqcup_n r'_n \cdot r_n^U) = \bigsqcup_n G^E(r'_n) \cdot G^E(r_n)^U$$

assuming that $\bigsqcup_n r_n \cdot r_n^U = id$.

For the least fixed point of a continuous function f the chain $(f^n(\bot))_n$ was of interest. For a locally continuous and symmetric functor G the analogue is

$$CHAIN(G) = ((D_n)_n, (e_n)_n)$$

where D_0=U (the one-element cpo), $D_{n+1} = G^E(D_n)$, $e_0 = \bot$ and $e_{n+1} = G^E(e_n)$. This gives a chain in $\underline{\underline{CPOe}}$.

Theorem 2.2:4 (essentially /SmPl82,PloLN/).

Let $G: \underline{\underline{CPO2s}} \rightarrow \underline{\underline{CPO2s}}$ be a locally continuous and symmetric functor.

Let $(D,(r_n)_n)$ be a limiting cone in $\underline{\underline{CPOe}}$ of CHAIN(G). Then (D,Θ) with $\Theta = \bigsqcup_n r_{n+1} \cdot G^E(r_n)^U$ is an initial fixed point of $G^E: \underline{\underline{CPOe}} \to \underline{\underline{CPOe}}$. ///

Sketch of proof. We shall write CHAIN(G) = $((D_n)_n,(e_n)_n)$. The chain $((D_{n+1})_n,(e_{n+1})_n)$ has $(D,(r_{n+1})_n)$ as a limiting cone but since the chain equals G^E[CHAIN(G)] it follows from the previous lemma that also $(G^E(D),(G^E(r_n))_n)$ is a limiting cone. Hence the mediating embedding $\Theta: G^E(D) \to D$ is an isomorphism. It follows that (D,Θ) is a fixed point for G^E and by theorem 2.2:2 the formula for Θ follows.

Next let (D',Θ') be a fixed point and suppose the embedding $e: D \to D'$ satisfies $e \cdot \Theta = \Theta' \cdot G^E(e)$. Then $e \cdot r_0 = \bot$ and $e \cdot r_{n+1} = e \cdot \Theta \cdot G^E(r_n) = \Theta' \cdot G^E(e \cdot r_n)$ shows that $e = e \cdot id_D = \bigsqcup_n (e \cdot r_n) \cdot r_n^U$ is uniquely determined. Next a cone $(D',(r'_n)_n)$ for CHAIN(G) may be defined by $r'_0 = \bot$ and $r'_{n+1} = \Theta' \cdot G^E(r'_n)$. Let $r: D \to D'$ be the mediating morphism from the limiting cone to this cone. Then

$$\begin{aligned} r \cdot \Theta &= \bigsqcup_n r \cdot r_{n+1} \cdot G^E(r_n)^U \\ &= \bigsqcup_n r'_{n+1} \cdot G^E(r_n)^U \\ &= \Theta' \cdot \bigsqcup_n G^E(r'_n) \cdot G^E(r_n)^U \\ &= \Theta' \cdot G^E(r) \end{aligned}$$

follows by (the discussion after) the previous lemma. ///

Note that local continuity was only required in order to use the previous lemma.

In general there are many limiting cones and initial fixed points, although they are all "isomorphic". When talking about <u>the</u> limiting cone or <u>the</u> initial fixed point we refer to the constructions of 2.2:2 and 2.2:4.

Viewing the solution in other categories

We now move the solutions from $\underline{\underline{\underline{\underline{CPOe}}}}$ to $\underline{\underline{\underline{\underline{\underline{CPO2s}}}}}$ and certain sub-categories of $\underline{\underline{\underline{\underline{CPOs}}}}$. We begin with chains, cones and limiting cones. So let $\underline{E} = ((D_n)_n,(e_n)_n)$ be a chain in $\underline{\underline{\underline{\underline{CPOe}}}}$. It is already a chain in $\underline{\underline{\underline{\underline{CPOs}}}}$ and a chain $S(\underline{E})$ in $\underline{\underline{\underline{\underline{\underline{CPO2s}}}}}$ may be defined by

$$S(\underline{E}) = ((D_n)_n,((e_n,e_n^U))_n)$$

Next let $\underline{R} = (D,(r_n)_n)$ be a cone of $\underline{E}$ in $\underline{\underline{\underline{\underline{CPOe}}}}$. It is already a cone of $\underline{E}$ in $\underline{\underline{\underline{\underline{CPOs}}}}$ and a cone $S(\underline{R})$ in $\underline{\underline{\underline{\underline{\underline{CPO2s}}}}}$ may be defined by

$$S(\underline{R}) = (D,((r_n,r_n^U))_n)$$

It is convenient to write $\underline{e}_n = (e_n,e_n^U)$ and $\underline{r}_n = (r_n,r_n^U)$. Note that $\underline{e}_n^U = \underline{e}_n^R$ and $\underline{r}_n^U = \underline{r}_n^R$.

As analogues of theorem 2.2:2 we have:

Lemma 2.2:5. Let $\underline{E} = ((D_n)_n,(\underline{e}_n)_n)$ be a chain in $\underline{\underline{\underline{\underline{\underline{CPO2s}}}}}$ with $\underline{e}_n$ embeddings such that $\underline{e}_n^U = \underline{e}_n^R$. It always has a limiting cone in $\underline{\underline{\underline{\underline{\underline{CPO2s}}}}}$. A cone $\underline{R} = (D,(\underline{r}_n)_n)$ is limiting iff $\underline{r}_n$ are embeddings and $\bigsqcup_n \underline{r}_n \cdot \underline{r}_n^U = id$. Then the mediating morphism from $\underline{R}$ to $(D',(\underline{r}'_n)_n)$ is $\bigsqcup_n r'_n \cdot r_n^U$. It is possible to choose $\underline{R}$ such that $\underline{r}_n^U = \underline{r}_n^R$. ///

Lemma 2.2:6. Let $\underline{E} = ((D_n)_n,(e_n)_n)$ be a chain in $\underline{\underline{\underline{\underline{CPOs}}}}$ with e_n embeddings. It always has a limiting cone in $\underline{\underline{\underline{\underline{CPOs}}}}$. A cone $\underline{R}=(D,(r_n)_n)$ is limiting iff r_n are embeddings and $\bigsqcup_n r_n \cdot r_n^U = id$. Then the mediating morphism from $\underline{R}$ to $(D',(r'_n)_n)$ is $\bigsqcup_n r'_n \cdot r_n^U$. ///

The proofs are similar to that of 2.2:2 and are omitted. It follows that if $\underline{R}$ is a limiting cone for $\underline{E}$ in $\underline{\underline{\underline{\underline{CPOe}}}}$ then $\underline{R}$ is a limiting cone for $\underline{E}$ in $\underline{\underline{\underline{\underline{CPOs}}}}$ and $S(\underline{R})$ is a limiting cone for $S(\underline{E})$ in $\underline{\underline{\underline{\underline{\underline{CPO2s}}}}}$. When talking about <u>the</u> limiting cone for $S(\underline{E})$ we mean $S(\underline{R})$ where $\underline{R}$ is

the limiting cone for $\underline{E}$. This means that an embedding $\underline{r}_n$ of the limiting cone for $S(\underline{E})$ has $\underline{r}_n^R$ as its upper adjoint.

When giving the semantics of the bottom-level metalanguage it is convenient to be able to use other categories than $\underline{\underline{CPO}}$. This motivates defining a notion of admissible subcategory. A sub cpo-category $\underline{\underline{B}}$ of a cpo-category $\underline{\underline{C}}$ is a subcategory such that

i) the morphisms are partially ordered as in $\underline{\underline{C}}$,

ii) the sets of morphisms of $\underline{\underline{B}}$ may be viewed as admissible predicates upon the corresponding sets of morphisms in $\underline{\underline{C}}$.

Clearly a sub cpo-category is itself a cpo-category. An example is $\underline{\underline{ACC}}$ that is a sub cpo-category of $\underline{\underline{CPO}}$. When $\underline{\underline{B}}$ is a sub cpo-category of $\underline{\underline{C}}$ every chain in $\underline{\underline{Be}}$ (the subcategory of embeddings) is a chain in $\underline{\underline{Ce}}$ as well. An admissible subcategory $\underline{\underline{B}}$ of a cpo-category $\underline{\underline{C}}$ is a sub cpo-category that

i) contains U,

ii) for every chain in $\underline{\underline{Be}}$ and limiting cone $\underline{R}$ in $\underline{\underline{Ce}}$ that $\underline{R}$ is a cone in $\underline{\underline{Be}}$.

(It will not do if $\underline{R}$ is only in $\underline{\underline{B}}$.) A further analogue of theorem 2.2:2 then is:

Lemma 2.2:7. Let $\underline{\underline{B}}$ be an admissible subcategory of $\underline{\underline{CPOs}}$ and $\underline{E}$ a chain in $\underline{\underline{Be}}$. The limiting cone calculated in $\underline{\underline{CPOe}}$ (by theorem 2.2:2) is also limiting in $\underline{\underline{B}}$ and the mediating morphism is as in $\underline{\underline{CPOs}}$. ///

Proof The mediating morphism from $\underline{\underline{CPOs}}$ is in $\underline{\underline{B}}$ because $\underline{\underline{B}}$ is a sub cpo-category of $\underline{\underline{CPOs}}$. This proves existence and uniqueness is because a cone in $\underline{\underline{B}}$ is also a cone in $\underline{\underline{CPOs}}$. ///

Example The category ACCs is an admissible subcategory of CPOs. To see this let $(D,(r_n)_n)$ be a limiting cone in CPOe for an ACCe chain $((D_n)_n,(e_n)_n)$. We first show that D is algebraic with $B_D = \{r_n(b_n) \mid b_n \in B_{D_n} \wedge n \geq 0\}$ and fact 2.2:1 will be used for this. If r is an embedding and b is finite then also r(b) is finite: for $r(b) \sqsubseteq \bigsqcup_n d_n$ gives $b \sqsubseteq \bigsqcup_n r^U(d_n)$ so there exists an n such that $b \sqsubseteq r^U(d_n)$ and $r(b) \sqsubseteq d_n$ follows. Next if $d \in D$ we have $d = \bigsqcup_n r_n(r_n^U(d))$ and for each n there is a chain $(b_m^n)_m$ of elements of B_{D_n} such that $r_n^U(d) = \bigsqcup_m b_m^n$. It follows that

$$d = \bigsqcup\{r_n(b_m^n) \mid n \geq 0 \wedge m \geq 0\}$$

To see D is consistently complete let H be a subset and d an upper bound. Then $\{r_n^U(h) \mid h \in H\}$ has $r_n^U(d)$ as an upper bound so $d'_n = \bigsqcup\{r_n^U(h) \mid h \in H\}$ exists. Also $(r_n(d'_n))_n$ is an increasing chain so $\bigsqcup_n r_n(d'_n)$ exists. It is the least upper bound of H in D. ///

We now move the fixed points across to CPO2s and admissible subcategories of CPOs. Since CPOs is an admissible subcategory of CPOs this includes CPOs as well. A generalisation of fixed points leads to:

Definition An algebra of a covariant functor G upon some category is a pair (D,g) where D is an object and $g:G(D) \rightarrow D$ is a morphism. The algebra is initial iff for any algebra (D',g') there is precisely one morphism $g'':D \rightarrow D'$ such that $g'' \cdot g = g' \cdot G(g'')$. ///

Clearly every fixed point is an algebra.

Theorem 2.2:8 (essentially /PloLN,SmP182/).

(1) Let G be a locally continuous and symmetric functor over CPO2s

with (D,Θ) the initial fixed point of G^E. Then $(D,(\Theta,\Theta^{-1}))$ is an initial algebra of G and an initial fixed point of G (so that the mediating morphism to another fixed point is an embedding).

(2) Let $\underline{\underline{B}}$ be an admissible subcategory of $\underline{\underline{CPO2s}}$ and F a locally continuous and covariant functor over $\underline{\underline{B}}$. Then F has an initial algebra that is also an initial fixed point and it may be constructed as in theorem 2.2:4. ///

The proof is analogous to that of 2.2:4 and is omitted.

Extending the solution to a functor

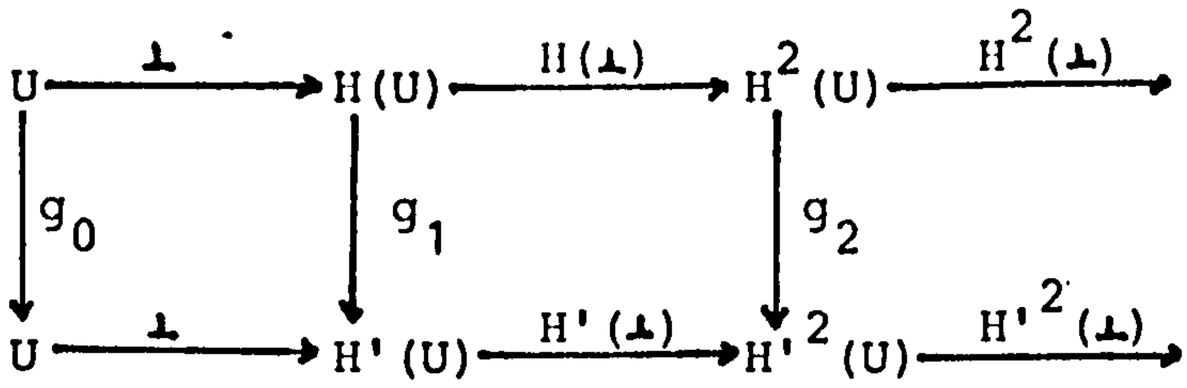
The metalanguage allows to nest recursive domain equations. The theory developed so far transforms a domain equation like $recX_i.X_1+...+X_N+A$ to a functor $G:\underline{\underline{CPO2s}}^N \rightarrow \underline{\underline{CPO2s}}$ in order to solve the equation. It is therefore necessary to formulate the solution as a functor.

We begin with considering a locally continuous and symmetric functor $G:\underline{\underline{CPO2s}}^N \rightarrow \underline{\underline{CPO2s}}$. The aim is to define a similar functor $REC_i(G):\underline{\underline{CPO2s}}^{N-1} \rightarrow \underline{\underline{CPO2s}}$. We shall subsequently assume i=1 and omit the subscript. It is straightforward to define the effect of REC(G) upon objects B_2, ..., B_N. Define $H = G^{\cdot}(Id,K_{B_2},...,K_{B_N})$ and note that it is a locally continuous and symmetric functor over $\underline{\underline{CPO2s}}$. It therefore has the initial fixed point (D,Θ) and we define $REC(G)(B_2,...,B_N) = D$.

To define the effect upon morphisms we consider the chains involved in constructing (D,Θ). The chain $\underline{E} = S(CHAIN(H))$ will be written $((H^n(U))_n,(H^n(\perp))_n)$ and let the limiting cone $\underline{R}$ in $\underline{\underline{CPO2s}}$

be $(D,(r_n)_n)$. We then have $\Theta = \bigsqcup_n r_n \cdot H(r_n)^U$. Let $f_i:B_i \to B_i'$ be morphisms and define $H' = G^*(Id,K_{B_2'},\ldots,K_{B_N'})$, $\underline{E}'$, $\underline{R}'$ and (D',Θ') similarly to above. To obtain a morphism from D to D' we will modify $\underline{R}'$ to a cone $\underline{Rg}'$ of $\underline{E}'$.

For this purpose define a grid from $\underline{E}$ to $\underline{E}'$ as a sequence $(g_n)_n$ where each $g_n:H^n(U) \to H'^n(U)$ is a $\underline{\underline{\text{CPO2s}}}$ morphism such that $g_{n+1} \cdot H^n(\perp) = H'^n(\perp) \cdot g_n$. This may be illustrated by



and the information that all small (hence all) rectangles commute. Now define $g_o = \perp$ and $g_{n+1} = G(g_n,f_2,\ldots,f_N)$. To see this defines a grid note that the leftmost square commutes (because we are in $\underline{\underline{\text{CPO2s}}}$) and since G is covariant this means that all others do as well. Now define $\underline{Rg}'$ as $(D',(r_n' \cdot g_n)_n)$ and note this is a cone on $\underline{E}$ because $(g_n)_n$ is a grid from $\underline{E}$ to $\underline{E}'$ and $\underline{R}'$ is a cone on $\underline{E}'$. By 2.2:5 there is precisely one mediating morphism r from $\underline{R}$ to $\underline{Rg}'$. It is $r = \bigsqcup_n r_n' \cdot g_n \cdot r_n^U$ (and $(r_n' \cdot g_n \cdot r_n^U)_n$ is a chain) and we define $REC(G) = r$.

We then have:

Theorem 2.2:9. If G is a locally continuous and symmetric functor over $\underline{\underline{\text{CPO2s}}}$ then so is REC(G). ///

Proof The identity functor law that $REC(G)(id,\ldots,id) = id$ is straightforward because each g_n is $id_{H^n(U)}$ and $id = \bigsqcup_n r_n \cdot r_n^U$ follows by 2.2:5. For the composition functor law let $f_i:B_i' \to B_i''$ and define H", $\underline{E}$", $\underline{R}$" as before. Let $(g_n)_n$ be the grid from $\underline{E}$ to $\underline{E}'$,

$(g'_n)_n$ the grid from $\underline{E}'$ to $\underline{E}''$ and $(g''_n)_n$ the grid from $\underline{E}$ to $\underline{E}''$. It is a straightforward numerical induction to show that $g''_n = g'_n \cdot g_n$. Then

$$\begin{aligned}
&\mathrm{REC}(G)(f'_2 \cdot f_2, \ldots, f'_N \cdot f_N) = \\
&\bigsqcup_n r''_n \cdot g'_n \cdot g_n \cdot r_n{}^U = \\
&\bigsqcup_n (r''_n \cdot g'_n \cdot r'_n{}^U) \cdot (r'_n \cdot g_n \cdot r_n{}^U) = \\
&\mathrm{REC}(G)(f'_2, \ldots, f'_N) \cdot \mathrm{REC}(G)(f_2, \ldots, f_N)
\end{aligned}$$

shows the result. To see that REC(G) is symmetric note that the grid from $\underline{E}'$ to $\underline{E}$ constructed for $f_2{}^R, \ldots, f_N{}^R$ is $(g_n{}^R)_n$. Then

$$\begin{aligned}
&\mathrm{REC}(G)(f_2{}^R, \ldots, f_N{}^R) = \\
&\bigsqcup_n (r_n \cdot g_n \cdot r'_n{}^U)^R = \\
&\mathrm{REC}(G)(f_2, \ldots, f_N)^R
\end{aligned}$$

follows because the cones $\underline{R}$ and $\underline{R}'$ have $r_n{}^U = r_n{}^R$ and $r'_n{}^U = r'_n{}^R$. Local continuity is immediate because the grid (i.e. each g_n) depends continuously on $f_2, \ldots, f_N$. ///

For later reference we state some lemmas about REC(G). The first amounts to another way of defining the effect upon morphisms.

Lemma 2.2:10. $\mathrm{REC}(G)(f_2, \ldots, f_N)$ is the least fixed point of the continuous function mapping a CPO2s morphism f to the morphism $\theta'' \cdot G(f, f_2, \ldots, f_N) \cdot \theta^{-1}$. ///

Proof Let G' denote the continuous function defined. It suffices to prove $G'^n(\bot) = r'_n \cdot g_n \cdot r_n{}^U$ by induction on n. The base case is immediate and

$$\begin{aligned}
G'^{n+1}(\bot) &= \theta' \cdot G(r'_n \cdot g_n \cdot r_n{}^U, f_2, \ldots, f_N) \cdot \theta^{-1} \\
&= \theta^{-1} \cdot H'(r'_n) \cdot g_{n+1} \cdot H(r_n{}^U) \cdot \theta^{-1} \\
&= r'_{n+1} \cdot g_{n+1} \cdot r_{n+1}{}^U
\end{aligned}$$

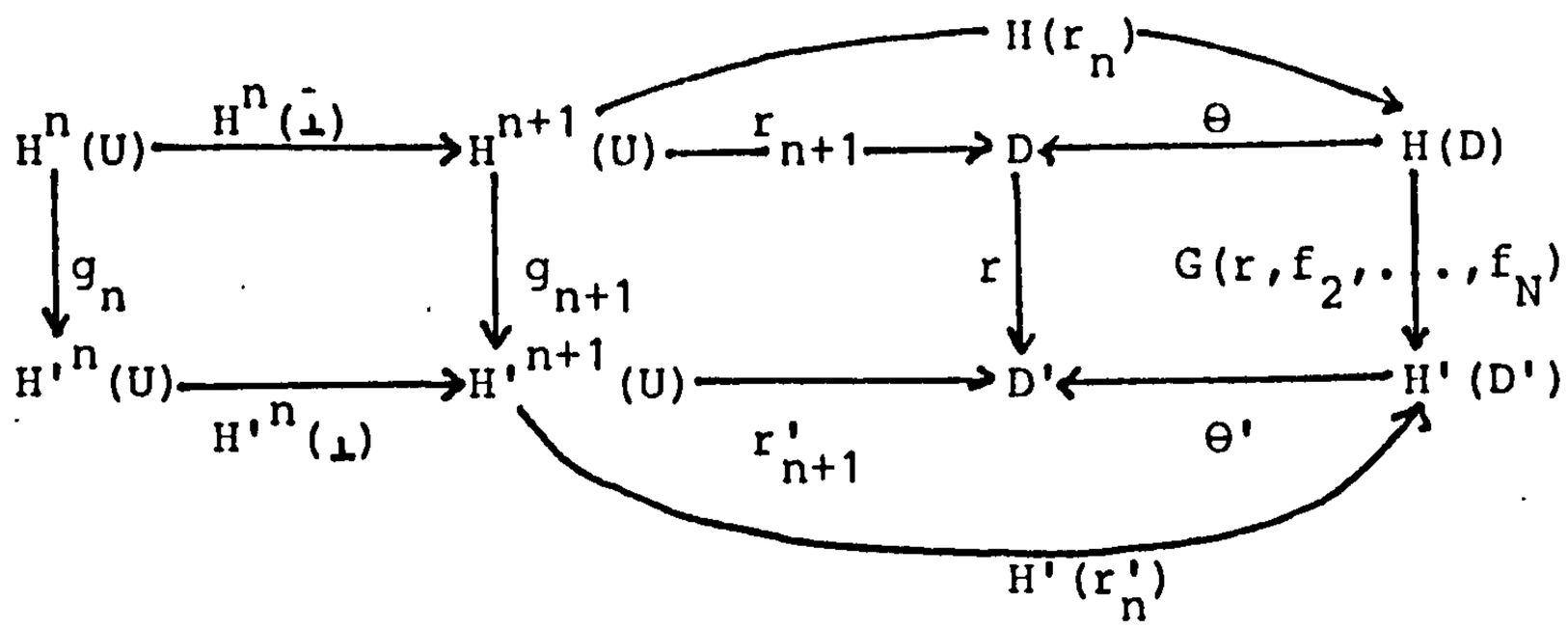
shows the inductive step. ///

Lemma 2.2:11. $r_n' \cdot g_n = REC(G)(f_2,\ldots,f_N) \cdot r_n$ and

$g_n \cdot r_n^U = r_n'^U \cdot REC(G)(f_2,\ldots,f_N)$ ///

Proof The first result follows from the definition of REC(G) and lemma 2.2:5. The second result follows from $g_n \cdot H^n(\perp)^U = H'^n(\perp)^U \cdot g_{n+1}$ using that $r_n'^U \cdot r_{n+1}' = H'^n(\perp)^U$. ///

It may help to summarise the last results by the following diagram



where $r = REC(G)(f_2,\ldots,f_N)$ and all polygons commute.

The development so far may be specialised to apply for locally continuous and covariant functors upon admissible subcategories of $\underline{\underline{CPOs}}$. So let $\underline{\underline{B}}$ be an admissible subcategory of $\underline{\underline{CPOs}}$ and F a locally continuous and covariant functor over $\underline{\underline{B}}$. Define H, H', $\underline{E}$, $\underline{E}'$, $\underline{R}$, $\underline{R}'$, (D,Θ), (D',Θ') and $(g_n)_n$ much as before and define

$$REC(F)(B_2,\ldots,B_N) = D$$

$$REC(F)(f_2,\ldots,f_N) = \bigsqcup_n r_n' \cdot g_n \cdot r_n^U$$

Lemma 2.2:12. If F is locally continuous and covariant over an admissible subcategory of $\underline{\underline{CPOs}}$ then also REC(F) is. ///

Analogues of lemmas 2.2:10 and 2.2:11 also hold in this setting.

Remark Suppose $F:\underline{\underline{CPOs}}^2 \to \underline{\underline{CPOs}}$ is a functor that is covariant in its left argument and contravariant in its right argument. Instead

of working with F^S over $\underline{\underline{CPO2s}}$ one could directly define F^{SE} over $\underline{\underline{CPOe}}$ by $F^{SE}(f,g) = F(f,g^U)$. This is what is done in /SmPl82/. Instead of working with local continuity one may use a categorical notion of continuity for covariant functors over $\underline{\underline{CPOe}}$. (Essentially 2.2:3 shows that if F is locally continuous then F^{SE} is continuous in the categorical sense.) This is done in /LeSm81/ where proofs have a distinct "categorical flavour". ///

...

2.3 SEMANTICS OF THE METALANGUAGE

Using the domain theory of section 2.2 we can now define the semantics of the metalanguage from section 2.1. As was sketched in the introduction (for basic blocks of flowcharts) it is of interest to assign several different meanings to the formulae of the metalanguage. To facilitate this we parameterise the semantics upon a specification of the details that may vary. Such a parameter will be called an interpretation. The idea is borrowed from algebra and logic and in data flow analysis similar ideas have motivated the monotone frameworks of /KaUl77/ and the interpretations of /CoCo77b/.

An interpretation consists of two parts: the type part and the expression part. We shall consider the type part before the expression part. For each of the two parts we begin with a formal definition. The definition is illustrated by defining that part of the standard interpretation, i.e. the interpretation that gives rise to the standard semantics /MiSt76/. Finally, the semantic equations for the corresponding part of the metalanguage is given.

TYPE PART

The type part of an interpretation $\underline{I}$ specifies information needed to define the functors $\underline{I}[\![t]\!]$ and $\underline{I}[\![gt]\!]$ that express the semantics of the types. A top-level type t will always be interpreted over $\underline{\underline{\underline{\underline{\underline{CPO2s}}}}}$, i.e. $\underline{I}[\![t]\!]$ will be a locally continuous and symmetric functor over $\underline{\underline{\underline{\underline{\underline{CPO2s}}}}}$. The need for morphisms to be pairs of functions has already been motivated (with contravariance of function space) as has the need for strictness (for + and $\otimes$). A subcategory of $\underline{\underline{\underline{CPO}}}$ (such as $\underline{\underline{\underline{ACC}}}$) could be used but no complications arise from the use of $\underline{\underline{\underline{CPO}}}$ and the larger category is therefore preferred. Domain constructors in the top-level will be as in section 2.2.

The bottom-level metalanguage will vary considerably in the way it is interpreted, so most constructs will be defined in the interpretation. There is no function space construction in the bottom-level metalanguage so $\underline{I}[\![gt]\!]$ may be interpreted as a locally continuous and covariant functor upon $\underline{\underline{\underline{\underline{CPOs}}}}$ (i.e. no need for $\underline{\underline{\underline{\underline{\underline{CPO2s}}}}}$). The bottom-level type structure will be interpreted in different settings so it is helpful to allow admissible subcategories of $\underline{\underline{\underline{\underline{CPOs}}}}$. Domain constructors are then defined as functors over that subcategory.

Formally the type part of an interpretation $\underline{I}$ is a tuple with the following three components:

- a sub cpo-category $\underline{\underline{B}}$ of $\underline{\underline{\underline{CPO}}}$ and a (necessarily admissible) predicate p upon each set of morphisms of $\underline{\underline{B}}$ such that $\underline{\underline{Bp}}$ (having as morphisms those of $\underline{\underline{B}}$ that satisfy p) is an admissible subcategory of $\underline{\underline{\underline{\underline{CPOs}}}}$,

- for each of the domain constructors $\underline{\times}$, $\underline{*}$, $\underline{+}$, $\underline{\perp}$ a locally continuous and covariant functor $\underline{\underline{Bp}}^k \rightarrow \underline{\underline{Bp}}$,
- for each bottom-level domain $\underline{A}_i$ a constant functor $\underline{\underline{Bp}}^0 \rightarrow \underline{\underline{Bp}}$.

We write $\underline{I}(\underline{\underline{B}})$, $\underline{I}(\underline{+})$, $\underline{I}(\underline{A}_i)$ etc. for these entities. We will assume throughout that the meaning of domains in the top-level metalanguage are fixed and therefore it has not been made part of "an interpretation".

...

The standard interpretation $\underline{S}$ has $\underline{S}(\underline{\underline{B}}) = \underline{\underline{\underline{ACC}}}$ and $\underline{S}(\underline{\underline{Bp}}) = \underline{\underline{\underline{\underline{ACCs}}}}$. It is helpful later (when constructing the powerdomain and the tensor product) that objects are algebraic. The assumption about consistently completeness is not necessary but would be needed if algebraicity had been assumed in the top-level type structure. The functors $\underline{S}(\underline{\times})$, $\underline{S}(\underline{*})$, $\underline{S}(\underline{+})$, and $\underline{S}(\underline{\perp})$ are as in section 2.2 but specialised to $\underline{\underline{\underline{\underline{ACCs}}}}$. Finally, $\underline{S}(\underline{A}_i):\underline{\underline{Bp}}^0 \rightarrow \underline{\underline{Bp}}$ will not be specified but one may assume they give the same objects as is given by A_i in the top-level.

For semantics more precision is needed about the connection between a domain variable X of t and the corresponding argument position for $\underline{I}[\![t]\!]$. We assume there is a bijective correspondence between the domain variables and the natural numbers. A domain variable X is said to have index i in the finite set V of domain variables iff exactly i-1 domain variables from V have an associated number less than that of X. Let card(V) denote the cardinality of V. If $V \vdash t$ we define a functor $\underline{I}[\![t]\!]_V:\underline{\underline{\underline{\underline{\underline{CPO2s}}}}}^{card(V)} \rightarrow \underline{\underline{\underline{\underline{\underline{CPO2s}}}}}$ with $\underline{I}[\![X]\!]_V = P_i$ (the i'th projection functor). Similarly $V \vdash gt$ gives $\underline{I}[\![gt]\!]_V:\underline{\underline{Bp}}^{card(V)} \rightarrow \underline{\underline{Bp}}$. The use of index V could be alleviated if it was always assumed that it was the least set V' such that $V' \vdash t$.

However, we shall mostly omit the subscript V and assume that the index is i=1.

For the details of the semantics of the top-level types:

$$\underline{I}[\![A_i]\!]_V = K_{A_i}$$

where $K_{A_i}: \underline{\underline{CPO2s}}^{card(V)} \rightarrow \underline{\underline{CPO2s}}$ is the functor that constantly gives the cpo A_i, and the A_i corresponding to T gives $T_\perp$

$$\underline{I}[\![t_1 \times \ldots \times t_k]\!]_V = \times^S \cdot (\underline{I}[\![t_1]\!]_V, \ldots, \underline{I}[\![t_k]\!]_V)$$

for $\times: \underline{\underline{CPOs}}^k \rightarrow \underline{\underline{CPOs}}$ defined in section 2.2; it is made symmetric and composed with the arguments

$$\underline{I}[\![t_1 * \ldots * t_k]\!]_V = *^S \cdot (\underline{I}[\![t_1]\!]_V, \ldots, \underline{I}[\![t_k]\!]_V)$$

$$\underline{I}[\![t_1 + \ldots + t_k]\!]_V = +^S \cdot (\underline{I}[\![t_1]\!]_V, \ldots, \underline{I}[\![t_k]\!]_V)$$

$$\underline{I}[\![t_\perp]\!]_V = (\,)_\perp^S \cdot \underline{I}[\![t]\!]_V$$

$$\underline{I}[\![t_1 \rightarrow t_2]\!]_V = \rightarrow^S \cdot (\underline{I}[\![t_1]\!]_V, \underline{I}[\![t_2]\!]_V)$$

where $\rightarrow$ is the function space construction of section 2.2

$$\underline{I}[\![\text{rec}X.t]\!]_V = \text{REC}_i(\,\underline{I}[\![t]\!]_{V \cup \{X\}}\,) \cdot (P_1, \ldots, P_{i-1}, P_{i+j}, \ldots, P_{card(V)})$$

where i is the index of X in $V \cup \{X\}$ and j=1 if $X \in V$ and otherwise j=0

$$\underline{I}[\![X]\!]_V = P_i$$

where i is the index of X in V and P_i is the i'th projection functor over $\underline{\underline{CPO2s}}$

$$\underline{I}[\![gt_1 \underline{\rightarrow} gt_2]\!]_V = (\underline{\rightarrow} \cdot (\underline{I}[\![gt_1]\!]_\emptyset, \underline{I}[\![gt_2]\!]_\emptyset) \cdot K_{NIL})^S$$

The last equation requires some explanation. Here $\underline{\rightarrow}$ is a mixed covariant and contravariant functor $\underline{\underline{Bp}}^2 \rightarrow \underline{\underline{CPOs}}$ giving on objects the cpo of $\underline{\underline{B}}$ morphisms from one to another and is defined in section 2.2 on morphisms. So $\underline{\rightarrow} \cdot (\underline{I}[\![gt_1]\!]_\emptyset, \underline{I}[\![gt_2]\!]_\emptyset)$ is a covariant functor $\underline{\underline{Bp}}^0 \rightarrow \underline{\underline{CPOs}}$. The category $\underline{\underline{Bp}}^0$ has one object NIL and composition with the constant functor $K_{NIL}: \underline{\underline{CPOs}}^{card(V)} \rightarrow \underline{\underline{Bp}}^0$ gives a functor

$\underline{\underline{CPOs}}^{card(V)} \rightarrow \underline{\underline{CPOs}}$ which is then made symmetric.

For the details of the semantics of the bottom-level types:

$$\underline{I}[\![\underline{A}_i]\!]_V = \underline{I}(\underline{A}_i) \cdot K_{NIL}$$

where $K_{NIL}: \underline{\underline{Bp}}^{card(V)} \rightarrow \underline{\underline{Bp}}^0$

$$\underline{I}[\![gt_1 \underline{x} \ldots \underline{x} gt_k]\!]_V = \underline{I}(\underline{x}) \cdot (\underline{I}[\![gt_1]\!]_V, \ldots, \underline{I}[\![gt_k]\!]_V)$$

$$\underline{I}[\![gt_1 \underline{*} \ldots \underline{*} gt_k]\!]_V = \underline{I}(\underline{*}) \cdot (\underline{I}[\![gt_1]\!]_V, \ldots, \underline{I}[\![gt_k]\!]_V)$$

$$\underline{I}[\![gt_1 \underline{+} \ldots \underline{+} gt_k]\!]_V = \underline{I}(\underline{+}) \cdot (\underline{I}[\![gt_1]\!]_V, \ldots, \underline{I}[\![gt_k]\!]_V)$$

$$\underline{I}[\![gt_{\underline{\perp}}]\!]_V = \underline{I}(\underline{\perp}) \cdot \underline{I}[\![gt]\!]_V$$

$$\underline{I}[\![\underline{rec} X.gt]\!]_V = REC_i(\underline{I}[\![gt]\!]_{V \cup \{X\}}) \cdot (P_1, \ldots, P_{i-1}, P_{i+j}, \ldots, P_N)$$

where i is the index of X in $V \cup \{X\}$, N is card(V) and

j=1 if $X \in V$ and otherwise j=0

$$\underline{I}[\![X]\!]_V = P_i$$

where i is the index of X in V

These definitions satisfy:

Lemma 2.3:1. If $V \vdash gt$ then $\underline{I}[\![gt]\!]_V: \underline{\underline{Bp}}^{card(V)} \rightarrow \underline{\underline{Bp}}$ is a locally continuous and covariant functor. If $V \vdash t$ then $\underline{I}[\![t]\!]_V: \underline{\underline{CPO2s}}^{card(V)} \rightarrow \underline{\underline{CPO2s}}$ is a locally continuous and symmetric functor. ///

The proof is straightforward by structural induction. It uses the properties of an interpretation $\underline{I}$ and the results of section 2.2.

For later reference it is convenient to state a consequence of the structural definition of $\underline{I}[\![t]\!]$ and $\underline{I}[\![gt]\!]$. Recall the syntactic substitutions $\ldots[\ldots/\ldots]$ mentioned in section 2.1.

Lemma 2.3:2. If $V \vdash t$, $V \vdash t'$, card(V) = N and X has index 1 in V then for all interpretions $\underline{I}$

$$\underline{I}[\![t[t'/X]]\!] = \underline{I}[\![t]\!] \cdot (\underline{I}[\![t']\!], P_2, \ldots, P_N)$$

and if additionally $V \vdash gt$, $V \vdash gt'$ then

$$\underline{I}[\![gt[gt'/X]]\!] = \underline{I}[\![gt]\!] \cdot (\underline{I}[\![gt']\!], P_2, \ldots, P_N) \qquad ///$$

Proof The proof is by structural induction on gt and t and let us restrict attention to the case $gt = \underline{rec}Y.gt_0$. When Y is X the result is straightforward so assume otherwise. Assume that Y is an element of V and has index N and that Y is not free in gt'. It is straightforward to adapt the proof should any of the assumptions fail (if Y is free in gt' a renaming with some Y' not free in gt' may be performed). Abbreviate the two sides of the equation to LHS and RHS respectively.

To show that $LHS(L_1,\ldots,L_N) = RHS(L_1,\ldots,L_N)$ define functors

$$LH = \underline{I}[\![gt_0]\!] \cdot (\underline{I}[\![gt']\!], P_2, \ldots, P_N) \cdot (K_{L_1}, \ldots, K_{L_{N-1}}, Id)$$

$$RH = \underline{I}[\![gt_0]\!] \cdot (K_{\underline{I}[\![gt']\!](L_1,\ldots,L_N)}, K_{L_2}, \ldots, K_{L_{N-1}}, Id)$$

It follows from the inductive hypothesis that $LHS(L_1,\ldots,L_N)$ is the object of the limiting cone $(L,(r_n)_n)$ for the chain $((LH^n(U))_n,(LH^n(\perp))_n)$ and similarly that $RHS(L_1,\ldots,L_N)$ is the object of the limiting cone $(\bar{L},(\bar{r}_n)_n)$ for $((RH^n(U))_n,(RH^n(\perp))_n)$. To show $L=\bar{L}$ and $r_n=\bar{r}_n$ it suffices to prove that $LH^n(U) = RH^n(U)$ and $LH^n(\perp) = RH^n(\perp)$ for all n. This may be proved inductively with n=0 obvious. For the inductive step one may calculate

$$LH^{n+1}(U) =$$

$$\underline{I}[\![gt_0]\!] (\underline{I}[\![gt']\!](L_1,\ldots,L_{N-1},LH^n(U)),L_2,\ldots,L_{N-1},LH^n(U)) =$$

(because Y is not free in gt')

$$\underline{I}[\![gt_0]\!](\underline{I}[\![gt']\!](L_1,\ldots,L_N),L_2,\ldots,L_{N-1},RH^n(U)) =$$

$$RH^{n+1}(U)$$

and $LH^n(\perp) = RH^n(\perp)$ is shown similarly.

To show $LHS(f_1,\ldots,f_N) = RHS(f_1,\ldots,f_N)$ note that we have the formula $LHS(f_1,\ldots,f_N) = \bigsqcup_n r'_n \cdot LS_n \cdot r_n^U$ when $LS_0 = \bot$ and

$$LS_{n+1} = \underline{I}[\![gt_0]\!](\underline{I}[\![gt']\!](f_1,\ldots,f_{N-1},LS_n),f_2,\ldots,f_{N-1},LS_n)$$

Similarly $RHS(f_1,\ldots,f_N) = \bigsqcup_n r'_n \cdot RS_n \cdot r_n^U$ when $RS_0 = \bot$ and

$$RS_{n+1} = \underline{I}[\![gt_0]\!](\underline{I}[\![gt']\!](f_1,\ldots,f_N),f_2,\ldots,f_{N-1},RS_n)$$

It is a consequence of what was shown earlier that the embeddings r'_n and r_n are the same in the two definitions. As before an inductive proof shows that $LS_n = RS_n$ and thereby $LHS(f_1,\ldots,f_N) = RHS(f_1,\ldots,f_N)$. ///

It is also convenient to state a lemma showing when two interpretations give the same semantics (use V'=V below). This furthermore shows the first use of the predicate P(V,t) defined in section 2.1.

Lemma 2.3:3. Let $V \vdash t$ and $P(V',t)$ and $card(V) = N$ and $card(V') = n$. Suppose V' is the subset of V of variables having index at most n in V. For any two interpretations $\underline{I}$ and $\underline{J}$ and cpo's D_i and E_i the two functors (over CPO2s)

$$\underline{I}[\![t]\!] \cdot (K_{D_1},\ldots,K_{D_n},P_{n+1},\ldots,P_N)$$

and

$$\underline{J}[\![t]\!] \cdot (K_{E_1},\ldots,K_{E_n},P_{n+1},\ldots,P_N)$$

are equal. ///

Proof The proof is by structural induction and only the case recX.t is non-trivial. This case follows the same lines as the previous proof and is therefore omitted. ///

EXPRESSION PART

The type t of an expression e is a top-level type with no free domain variables. More precisely there is a type environment tenv with finite domain dom(tenv) such that tenv⊢e:t holds and we assumed in section 2.1 that t and all tenv(x) were closed. The semantics of the type therefore is a constant functor and it is the object that is of interest. This motivates writing $\underline{I}[\![t]\!]$ instead of $\underline{I}[\![t]\!]_{\emptyset}(NIL)$. With this notation the semantics of e is a function

$$\underline{I}[\![e]\!]_{tenv}:(\prod_{x \in dom(tenv)} \underline{I}[\![tenv(x)]\!]) \rightarrow \underline{I}[\![t]\!]$$

The function is continuous but since all morphisms encountered in chapters 2 to 4 will be continuous we shall often let the statement of continuity be implicit. Also we shall shortly omit the subscript tenv when confusion is not likely to occur.

The top-level aspects of the metalanguage will always be the same. The bottom-level aspects are defined in the expression part of an interpretation. Formally it is a tuple with three components:

- for each of the symbols $take_i$, $smashtake_i$, in_i, up, fold and unfold some functions must be specified. Taking $take_i$ as an example it is assumed that tenv⊢$take_i$:ft and the function must be an element of $\underline{I}[\![ft]\!]$ (and is hence continuous). This is slightly imprecise because $take_i$ has not been indexed with its type ft.

- for each of the symbols tuple, smashtuple, case, lift, cond and ◻ some functional must be specified, i.e. an element of

$$\underline{I}[\![ft_1]\!] \times \ldots \times \underline{I}[\![ft_n]\!] \rightarrow \underline{I}[\![ft]\!]$$

Taking tuple as an example it is required that

$\bigwedge_{i=1}^{n}$ tenv⊢ e_i:ft_i implies tenv⊢ tuple e_1,...,e_n:ft

and again this is slightly imprecise because tuple has not been indexed with ft_1, ..., ft_n, ft.

- for each constant f_i of contravariantly pure type t_i there must be an element of $\underline{I}[\![t_i]\!]$.

We write $\underline{I}(take_i)$, $\underline{I}(tuple)$, $\underline{I}(f_i)$ etc. for these entities. The conditions upon the functionals are essentially as in the notion of a "continuous algebra".

To define the expression part of the standard interpretation $\underline{S}$ we use the conditional $v_1 \to v_2, v_3$; it denotes v_2, v_3 or $\perp$ according to whether v_1 is tt, ff or $\perp$. The functions are given by:

$$\underline{S}(take_i)(v) = v\downarrow i$$

$$\underline{S}(smashtake_i)(v) = v\Downarrow i$$

$$\underline{S}(in_i)(v) = in_i(v)$$

$$\underline{S}(up)(v) = up(v)$$

$$\underline{S}(fold)(v) = \Theta(v) \qquad \text{-- see below}$$

$$\underline{S}(unfold)(v) = \Theta^{-1}(v) \qquad \text{-- see below}$$

Here it is assumed that $\emptyset \vdash fold: gt[\underline{rec}X.gt/X] \to \underline{rec}X.gt$ and that Θ is the second component of the initial fixed point (D,Θ) of the functor $\underline{I}[\![gt]\!]_{\{X\}}: \underline{\underline{ACCs}} \to \underline{\underline{ACCs}}$. (That the functionalities of Θ and Θ^{-1} are correct follows from lemma 2.3:2.) The functionals are given by:

$$\underline{S}(tuple)(f_1,...,f_k)(v) = (f_1(v),...,f_k(v))$$

$$\underline{S}(smashtuple)(f_1,...,f_k)(v) = smash(f_1(v),...,f_k(v))$$

$$\underline{S}(case)(f_1,...,f_k)(v) = is_1(v) \to f_1(out_1(v)),\ (\ is_2(v) \to f_2(out_2(v)),\ \ldots\ is_k(v) \to f_k(out_k(v)), \perp)$$

$$\underline{S}(\text{lift})(f)(v) = \text{def}(v) \rightarrow f(\text{down}(v)), \perp$$

$$\underline{S}(\text{cond})(f_1,f_2,f_3)(v) = f_1(v) \rightarrow f_2(v), f_3(v)$$

$$\underline{S}(\square)(f_1,f_2)(v) = f_1(f_2(v))$$

Finally, $\underline{S}(f_i)$ will not be specified. It is straightforward to verify that $\underline{S}$ is an interpretation.

The definition of the semantics is mostly straightforward but for completeness sake we give all the clauses. We begin with the top-level metalanguage and use the notation of the previous section:

$$\underline{I}[\![(e_1,\ldots,e_k)]\!]_{tenv}(env) = (\underline{I}[\![e_1]\!]_{tenv}(env),\ldots,\underline{I}[\![e_k]\!]_{tenv}(env))$$

$$\underline{I}[\![e'\downarrow i]\!]_{tenv}(env) = (\underline{I}[\![e']\!]_{tenv}(env))\downarrow i$$

$$\underline{I}[\![(*e_1,\ldots,e_k*)]\!]_{tenv}(env) = \text{smash}(\underline{I}[\![e_1]\!]_{tenv}(env),\ldots))$$

$$\underline{I}[\![e'\downarrow i]\!]_{tenv}(env) = (\underline{I}[\![e']\!]_{tenv}(env))\downarrow i$$

$$\underline{I}[\![\text{is}_i\ e']\!]_{tenv}(env) = \text{is}_i(\underline{I}[\![e']\!]_{tenv}(env))$$

$$\underline{I}[\![\text{in}_i\ e']\!]_{tenv}(env) = \text{in}_i(\underline{I}[\![e']\!]_{tenv}(env))$$

$$\underline{I}[\![\text{out}_i\ e']\!]_{tenv}(env) = \text{out}_i(\underline{I}[\![e']\!]_{tenv}(env))$$

$$\underline{I}[\![\text{def}\ e']\!]_{tenv}(env) = \text{def}(\underline{I}[\![e']\!]_{tenv}(env))$$

$$\underline{I}[\![\text{up}\ e']\!]_{tenv}(env) = \text{up}(\underline{I}[\![e']\!]_{tenv}(env))$$

$$\underline{I}[\![\text{down}\ e']\!]_{tenv}(env) = \text{down}(\underline{I}[\![e']\!]_{tenv}(env))$$

$$\underline{I}[\![\lambda x{:}t.e']\!]_{tenv}(env) = f$$

where $f(v) = \underline{I}[\![e']\!]_{tenv[t/x]}(env[v/x])$

$$\underline{I}[\![e'(e'')]\!]_{tenv}(env) = (\underline{I}[\![e']\!]_{tenv}(env))(\underline{I}[\![e'']\!]_{tenv}(env))$$

$$\underline{I}[\![x]\!]_{tenv}(env) = env(x)$$

$$\underline{I}[\![\text{mkrec}\ e']\!]_{tenv}(env) = \Theta(\underline{I}[\![e']\!]_{tenv}(env))$$

where $tenv \vdash \text{mkrec}\ e' : \text{rec}X.t$ and $(D,(\Theta,\Theta^{-1}))$ is the initial fixed point of $\underline{I}[\![t]\!]$ over $\underline{\underline{\text{CPO2s}}}$; by lemma 2.3:2 the functionality is correct

$$\underline{I}[\![\text{unrec}\ e']\!]_{tenv}(env) = \Theta^{-1}(\underline{I}[\![e']\!]_{tenv}(env))$$

where $tenv \vdash e' : \text{rec}X.t$ and Θ^{-1} is as above

$$\underline{I}[\![e_1 \rightarrow e_2, e_3]\!]_{tenv}(env) = \underline{I}[\![e_1]\!]_{tenv}(env) \rightarrow \underline{I}[\![e_2]\!]_{tenv}(env), \underline{I}[\![e_3]\!]_{tenv}(env)$$

$$\underline{I}[\![Y\ e']\!]_{tenv}(env) = LFP(\ \underline{I}[\![e']\!]_{tenv}(env)\ )$$

$$\underline{I}[\![f_i]\!]_{tenv}(env) = \underline{I}(f_i)$$

and for the bottom-level:

$$\underline{I}[\![tuple\ e_1,\ldots,e_k]\!]_{tenv}(env) = \underline{I}(tuple)(\underline{I}[\![e_1]\!]_{tenv}(env), \ldots)$$

$$\underline{I}[\![take_i]\!]_{tenv}(env) = \underline{I}(take_i)$$

$$\underline{I}[\![smashtuple\ e_1,\ldots,e_k]\!]_{tenv}(env) =$$

$$\underline{I}(smashtuple)(\underline{I}[\![e_1]\!]_{tenv}(env),\ldots,\underline{I}[\![e_k]\!]_{tenv}(env))$$

$$\underline{I}[\![smashtake_i]\!]_{tenv}(env) = \underline{I}(smashtake_i)$$

$$\underline{I}[\![case\ e_1,\ldots,e_k]\!]_{tenv}(env) = \underline{I}(case)(\underline{I}[\![e_1]\!]_{tenv}(env), \ldots)$$

$$\underline{I}[\![in_i]\!]_{tenv}(env) = \underline{I}(in_i)$$

$$\underline{I}[\![lift\ e']\!]_{tenv}(env) = \underline{I}(lift)(\underline{I}[\![e']\!]_{tenv}(env))$$

$$\underline{I}[\![up]\!]_{tenv}(env) = \underline{I}(up)$$

$$\underline{I}[\![fold]\!]_{tenv}(env) = \underline{I}(fold)$$

$$\underline{I}[\![unfold]\!]_{tenv}(env) = \underline{I}(unfold)$$

$$\underline{I}[\![cond\ e_1,e_2,e_3]\!]_{tenv}(env) = \underline{I}(cond)(\underline{I}[\![e_1]\!]_{tenv}(env), \ldots)$$

$$\underline{I}[\![e_1\ \alpha\ e_2]\!]_{tenv}(env) = \underline{I}(\alpha)(\underline{I}[\![e_1]\!]_{tenv}(env), \underline{I}[\![e_2]\!]_{tenv}(env))$$

Lemma 2.3:4. If $tenv \vdash e:t$ then the above equations define a continuous function $\underline{I}[\![e]\!]_{tenv}: \Pi_{x \in dom(tenv)} \underline{I}[\![tenv(x)]\!] \rightarrow \underline{I}[\![t]\!]$. ///

The proof is by structural induction and is omitted. It is a consequence that LFP is only applied to a continuous function.

The notion of interpretation defined here extends that of /CoCo77b/ in e.g. including specification of functionals rather than assuming a fixed interpretation of these. Still some things that are fixed might ideally be allowed to vary and this identifies limitations

in the present degree of "ambition". One such point is the fixed interpretation of $\underline{\rightarrow}$. Another is that Y and rec always give the least fixed point and initial solution, respectively. This will be discussed further in chapter 6.

Remark The use of $\underline{\underline{\underline{ACC}}}$ rather than $\underline{\underline{\underline{\underline{ACCs}}}}$ for $\underline{S}(\underline{\underline{B}})$ is not entirely satisfactory. An argument against $\underline{\underline{\underline{\underline{ACCs}}}}$ is that up is not strict. A stronger argument in favour of $\underline{\underline{\underline{\underline{ACCs}}}}$ is that the morphisms in the bottom-level express "state transformations" and they should be strict because "once something has failed to terminate nothing more can be done" /Sto77 p.203/. This will of course entail removing up (and lift and $\underline{\perp}$) and claim that up does not correspond to any operational intuition. This amounts to continuing the discussion started in the second remark in section 2.1. In view of this it is interesting to note that $\underline{\perp}$ gives "problems" in section 4.4 which are similar to those $\underline{\chi}$ gives. ///

2.4 APPLICATIONS OF THE METALANGUAGE

The notation for the bottom-level metalanguage may be somewhat unfamiliar and there are limitations in the types allowed. The purpose of this section is therefore to show that the metalanguage is still useable. This is done by giving two example semantics but they will not be needed later.

The first and major example is an imperative language with procedures. The abstract syntax of programs pro, expressions exp, commands cmd and declarations dcl is given by:

pro ::= program cmd

exp ::= num | true | false | read | ide | ide(exp_1,...,exp_k)
| if exp_1 then exp_2 else exp_3 | exp_1 ope exp_2

cmd ::= ide := exp | write exp | ide(exp_1,...,exp_k)
| if exp then cmd_1 else cmd_2 | while exp do cmd
| begin dcl ; cmd end | cmd_1;cmd_2

dcl ::= var ide := exp | fun ide(ide_1,...,ide_k); exp
| proc ide(ide_1,...,ide_k) | dcl_1;dcl_2

The syntax of identifiers ide, numerals num and operators ope is left unspecified. The language is similar to SMALL /Gor79/ but there are some syntactic deviations. For simplicity only identifiers may be assigned to, but to be general functions and procedures may have more than one argument. Variables are declared by var ide := exp but there is no facility for constants. The potential problem with constants is that we do not allow t::=gt. So one would have to treat constants using locations (as will be done for variables) or syntactic constraints must ensure that the identifiers referenced when initialising a constant are themselves constants. A semantic deviation is that we shall use call-by-value.

We now define the functionalities of the semantic valuations $\mathcal{P}$, $\mathcal{E}$, $\mathcal{C}$ and $\mathcal{D}$. The semantics will use locations and, to illustrate some possibilities, use a mixture of continuation style and direct style. To state the functionalities we need the bottom-level types I (input), O (output), S (state), E (value) and the top-level types L (locations) and R (environments). The functionalities to be used are listed below and explained afterwards.

$\mathcal{P}[\![pro]\!]$: $\underline{I \to O}$ direct style

$\mathcal{E}[\![exp]\!]$: $R \to L \to \underline{S \to E * S}$ direct style

$\mathcal{C}[\![cmd]\!]$: $R \to (L \to \underline{S \to S}) \to (L \to \underline{S \to S})$ continuation style

$\mathcal{D}[\![dcl]\!]$: $R \times L \to R \times L \times \underline{S \to S}$ direct style

Programs should be straightforward. For expressions one might have expected $\mathcal{E}[\![exp]\!]: R \to \underline{S \to E * S}$, i.e. that given an environment a "state transformation" is produced. However, the semantics of functions will require to know the "next free location" so that the arguments can be stored from there onwards. This information is available for the choice taken. Similar considerations apply for commands and declarations.

Before proceeding any further it is appropriate to point out that the metalanguage has been designed to be "minimal" in not having explicit notation for constructs that can be built from more primitive constructs. This simplifies the formal development but makes examples more cumbersome. An example is $\underline{M} = \underline{\text{rec}}M.\underline{0+E*M}$ that will be used to associate locations (in the form of natural numbers) with their values. An informal expansion of the definition gives $\underline{0+E*(0+...)}$ which may be rearranged to give $\underline{0+E+E^2+E^3+...}$. So $\underline{M}$ is the type of finite lists of values (of type $\underline{E}$) and location 1 may be viewed as corresponding to the 1'th element. Operations upon such lists can be defined using the constructs of the metalanguage but the definitions become rather detailed. An example of this is the function get below for accessing the contents of a location. One way to improve upon this would be to extend the metalanguage with a list forming constructor and the associated operations. A more ad hoc solution would be to assume that $\underline{M}$ is a base type and that the operations are constants (thereby side stepping the metalanguage). We shall not do

either as this section still serves its purpose without. But for this reason no "worked examples" about the behaviour of the semantics will be given.

Continuing with the development we define the domains as follows:

$\underline{I} = \underline{rec}I.\underline{\mathbb{O}+E\ast I}$	input
$\underline{O} = \underline{rec}O.\underline{\mathbb{O}+E\ast O}$	output
$\underline{M} = \underline{rec}M.\underline{\mathbb{O}+E\ast M}$	"memory"
$\underline{S} = \underline{I\ast M\ast O}$	states
$\underline{E} = \underline{T} + \ldots$	values
$\underline{T}$	truth values
$L = N$	locations
$F = L\to \underline{E\ast\ldots\ast E\ast S\to E\ast S}$	functions
$P = (L\to \underline{S\to S})\to (L\to \underline{E\ast\ldots\ast E\ast S\to S})$	procedures
$R = Ide\to L+F+P$	environments

The intention with these definitions is that e.g. $\underline{S}$ is a shorthand for $(\underline{rec}I.\underline{\mathbb{O}+E\ast I})\underline{\ast}\ldots\underline{\ast}(\underline{rec}O.\underline{\mathbb{O}+E\ast O})$.

Most of the details due to the "minimality" of the metalanguage are encapsulated by the following four auxiliary functions. An expression like $[\ldots]$ will be used instead of formally defining a constant function with that effect (in the standard semantics). Furthermore, where-clauses are used to help impose some structure upon the definitions. Also all occurrences of smashtuple and smash-$take_i$ are abbreviated to tuple and $take_i$. Finally, we shall write e.g. 1-1 as a shorthand for the more correct $[\lambda 1.1-1](1)$.

read : $\underline{S \to E * S}$ (reads the next input value)

read = tuple(cases($[\lambda x.\text{error value}]$, take_1) $\square$ unfold $\square$ take_1

, tuple(abbr_1, take_2, take_3))

$\underline{\text{where}}$ abbr_1 = cases(fold$\square$$\text{in}_1$, take_2) $\square$ unfold $\square$ take_1

write : $\underline{E * S \to S}$ (writes the value on the output)

write = tuple($\text{take}_1 \square \text{take}_2$, $\text{take}_2 \square \text{take}_2$,

... fold$\square$$\text{in}_2 \square$tuple($\text{take}_1$, $\text{take}_3 \square \text{take}_2$))

get : $L \to \underline{S \to E * S}$ (gets content of location)

get = $\lambda l{:}L.$ tuple($(\text{abbr}_1)(l) \square \text{take}_2$, $[\lambda s.s]$)

$\underline{\text{where}}$ abbr_1 = Y($\lambda g{:}L \to \underline{M \to E}.\lambda l{:}L.$ $\text{abbr}_2 \square$unfold)

$\underline{\text{where}}$ abbr_2 = cases($[\lambda i.\text{error value}]$

, (l=1 $\to$ take_1, g(l-1)$\square$$\text{take}_2$))

set: $L \to \underline{E * S \to S}$ (updates contents of location)

set = $\lambda l{:}L.$ tuple($\text{take}_1 \square \text{take}_2$, abbr_1, $\text{take}_3 \square \text{take}_2$)

$\underline{\text{where}}$ abbr_1 = $(\text{abbr}_2)(l) \square$tuple($\text{take}_1$, $\text{take}_2 \square \text{take}_2$)

$\underline{\text{where}}$ abbr_2 = Y($\lambda g{:}L \to \underline{E * M \to M}.\lambda l{:}L.$ abbr_3)

$\underline{\text{where}}$ abbr_3 = fold$\square$cases(abbr_7, abbr_6)$\square$ $[\text{abbr}_5] \square \text{abbr}_4$

$\underline{\text{where}}$ abbr_4 = tuple(take_1

, cases($\text{in}_1 \square$fold$\square$$\text{in}_1$, in_2)$\square$unfold$\square$$\text{take}_2$)

(of functionality $\underline{E * M \to E * (M + E * M)}$)

$\underline{\text{where}}$ $[\text{abbr}_5]$ = $[\lambda(x,y).$ $\text{is}_1(y) \to \text{in}_1((x, \text{out}_1(y)))$

, $\text{in}_2((x, \text{out}_2(y)))]$

(of functionality $\underline{E * (M + E * M) \to (E * M) + E * (E * M)}$)

$\underline{\text{where}}$ abbr_6 = $\text{in}_2 \square$(l=1 $\to$ tuple(take_1, $\text{take}_2 \square \text{take}_2$)

, tuple($\text{take}_1 \square \text{take}_2$,

g(l-1)$\square$tuple(take_1, $\text{take}_2 \square \text{take}_2$)))

(of functionality $\underline{E * (E * M) \to 0 + E * M}$)

$$\underline{\text{where}}\ \text{abbr}_7 = \text{in}_2\,\natural\,(1=1 \rightarrow [\lambda x.x]$$
$$, \text{tuple}([\lambda x.\ \text{some initial value}], g(1-1)))$$

(of functionality $\underline{E \times M \rightarrow \mathbb{O} + E \not\rightarrow M}$)

In the definition of get and set it should be noted that the fixed point operator Y has type $(t \rightarrow t) \rightarrow t$ for $t = L \rightarrow ft$. This shows an "interaction" between the levels of types that seems to have no analogue for the recursion operators rec and $\underline{\text{rec}}$ allowed for types.

The semantic functions are defined below, using the same notational conventions as above:

$$\mathcal{P}[\![\text{pro}]\!]: \underline{I \rightarrow O}$$

$$\mathcal{P}[\![\underline{\text{program}}\ \text{cmd}]\!] = \text{take}_3\,\natural$$
$$(\mathcal{C}[\![\text{cmd}]\!]\ [\text{some environment}]\ \lambda 1.[\lambda s.s]\ \ 1)\,\natural$$
$$\text{tuple}([\lambda i.i], [\lambda i.\text{some memory}], [\lambda i.\text{some output}])$$

The definition of expressions uses the notation m_k that is defined below:

$$\mathcal{E}[\![\text{exp}]\!]: R \rightarrow L \rightarrow \underline{S \rightarrow E * S}$$

$$\mathcal{E}[\![\text{num}]\!]\ r\ 1 = [\lambda s.((\text{some value}), s)]$$

$$\mathcal{E}[\![\underline{\text{true}}]\!]\ r\ 1 = [\lambda s.(\text{in}_1(\text{tt}), s)]$$

$$\mathcal{E}[\![\underline{\text{false}}]\!]\ r\ 1 = [\lambda s.(\text{in}_1(\text{ff}), s)]$$

$$\mathcal{E}[\![\underline{\text{read}}]\!]\ r\ 1 = \text{read}$$

$$\mathcal{E}[\![\text{ide}]\!]\ r\ 1 = \text{is}_1(r[\![\text{ide}]\!]) \rightarrow \text{get}(\text{out}_1(r[\![\text{ide}]\!])), [\lambda s.(\text{error value}, s)]$$

$$\mathcal{E}[\![\text{ide}(\text{exp}_1, \ldots, \text{exp}_k)]\!]\ r\ 1 = \text{is}_2(r[\![\text{ide}]\!]) \rightarrow \text{abbr}_1, [\lambda s.(\text{error}, s)]$$

$$\underline{\text{where}}\ \text{abbr}_1 = (\text{out}_2(r[\![\text{ide}]\!])(1))\,\natural\,m_{k-1}(\mathcal{E}[\![\text{exp}_k]\!]\ r\ 1)\,\natural$$
$$\ldots\,\natural(\mathcal{E}[\![\text{exp}_1]\!]\ r\ 1)$$

$$\mathcal{E}[\![\underline{\text{if}}\ \text{exp}_1\ \underline{\text{then}}\ \text{exp}_2\ \underline{\text{else}}\ \text{exp}_3]\!]\ r\ 1 = \text{cond}(\text{abbr}_1, \text{abbr}_2, \text{abbr}_3)$$

$$\underline{\text{where}}\ \text{abbr}_1 = [\text{out}_1]\,\natural\,\text{take}_1\,\natural(\mathcal{E}[\![\text{exp}_1]\!]\ r\ 1)$$

$$\underline{\text{where}}\ \text{abbr}_2 = (\mathcal{E}[\![\text{exp}_2]\!]\ r\ 1)\,\natural\,\text{take}_2\,\natural(\mathcal{E}[\![\text{exp}_1]\!]\ r\ 1)$$

$$\underline{\text{where}}\ \text{abbr}_3 = (\mathcal{E}[\![\text{exp}_3]\!]\ r\ 1)\,\natural\,\text{take}_2\,\natural(\mathcal{E}[\![\text{exp}_1]\!]\ r\ 1)$$

$$\mathcal{E}[\![\exp_1 \text{ ope } \exp_2]\!]\ r\ l = [\text{operator}] \natural m_1(\mathcal{E}[\![\exp_2]\!]\ r\ l) \natural \mathcal{E}[\![\exp_1]\!] r\ l$$

We now explain the equation for $ide(exp_1,\ldots,exp_k)$ when k=2. Then $\mathcal{E}[\![exp_1]\!]$ r l and $\mathcal{E}[\![exp_2]\!]$r l are of functionality $\underline{S \to E * S}$ and therefore cannot be composed. This is remedied by using

$$m_1(\mathcal{E}[\![\exp_2]\!]\ r\ l) : \underline{E * S \to E * E * S}$$

The idea is that the first argument remains unchanged. The formal definition is

$$m_k(f) = \text{tuple}(\text{take}_1,\ldots,\text{take}_k,$$
$$\text{take}_1 \natural f \natural \text{take}_{k+1},\ \text{take}_2 \natural f \natural \text{take}_{k+1})$$

The function m_k is not of contravariantly pure type so $m_k(f)$ should be regarded as a shorthand for the defining expression. (The definition of m_k may be compared with fit of /MiSt76/.)

For commands we have

$$\mathcal{E}[\![\text{cmd}]\!] : R \to (L \to \underline{S \to S}) \to (L \to \underline{S \to S})$$

$$\mathcal{E}[\![\text{ide} := \text{exp}]\!]\ r\ c\ l = c(l) \natural \text{abbr}_1 \natural (\mathcal{E}[\![\text{exp}]\!]\ r\ l)$$

$$\underline{\text{where}}\ \text{abbr}_1 = \text{is}_1(r[\![\text{ide}]\!]) \to \text{set}(\text{out}_1(r[\![\text{ide}]\!])), [\lambda(v,s).\text{error}]$$

$$\mathcal{E}[\![\underline{\text{write}}\ \text{exp}]\!]\ r\ c\ l = c(l) \natural \text{write} \natural (\mathcal{E}[\![\text{exp}]\!]\ r\ l)$$

$$\mathcal{E}[\![\text{ide}(\text{exp}_1,\ldots,\text{exp}_k)]\!]\ r\ c\ l = \text{is}_3(r[\![\text{ide}]\!]) \to \text{abbr}_1, [\lambda s.\text{error state}]$$

$$\underline{\text{where}}\ \text{abbr}_1 = (\text{out}_3(r[\![\text{ide}]\!])\ c\ l) \natural m_{k-1}(\mathcal{E}[\![\text{exp}_k]\!]\ r\ l) \natural$$
$$\ldots \natural (\mathcal{E}[\![\text{exp}_1]\!]\ r\ l)$$

$$\mathcal{E}[\![\underline{\text{if}}\ \text{exp}\ \underline{\text{then}}\ \text{cmd}_1\ \underline{\text{else}}\ \text{cmd}_2]\!]\ r\ c\ l =$$
$$\text{cond}([\text{out}_1] \natural \text{take}_1 \natural (\mathcal{E}[\![\text{exp}]\!]\ r\ l)$$
$$, (\mathcal{E}[\![\text{cmd}_1]\!]\ r\ c\ l) \natural \text{take}_2 \natural (\mathcal{E}[\![\text{exp}]\!]\ r\ l)$$
$$, (\mathcal{E}[\![\text{cmd}_2]\!]\ r\ c\ l) \natural \text{take}_2 \natural (\mathcal{E}[\![\text{exp}]\!]\ r\ l)$$

$$\mathcal{E}[\![\underline{\text{while}}\ \text{exp}\ \underline{\text{do}}\ \text{cmd}]\!]\ r\ c = Y(\lambda c' : L \to \underline{S \to S}.\lambda l : L.\ \text{abbr}_1)$$

$$\underline{\text{where}}\ \text{abbr}_1 = \text{cond}([\text{out}_1] \natural \text{take}_1 \natural \mathcal{E}[\![\text{exp}]\!]\ r\ l$$

$$,\mathcal{E}[\![\text{cmd}]\!]\ r\ c'\ 1 \sqsupset \text{take}_2 \sqsupset \mathcal{E}[\![\text{exp}]\!]\ r\ 1$$

$$,c(1) \sqsupset \text{take}_2 \sqsupset \mathcal{E}[\![\text{exp}]\!]\ r\ 1\ )$$

$$\mathcal{C}[\![\underline{\text{begin}}\ \text{dcl};\ \text{cmd}\ \underline{\text{end}}]\!]\ r\ c = (\mathcal{C}[\![\text{cmd}]\!]\ \text{abbr}_1\ c\ \text{abbr}_2) \sqsupset \text{abbr}_3$$

$$\underline{\text{where}}\ \text{abbr}_i = \mathcal{D}[\![\text{dcl}]\!](r,1)\downarrow i$$

$$\mathcal{C}[\![\text{cmd}_1;\text{cmd}_2]\!]\ r\ c = \mathcal{C}[\![\text{cmd}_1]\!]\ r\ (\mathcal{C}[\![\text{cmd}_2]\!]\ r\ c)$$

For declarations we need a variant of m_k:

$$m'_k(f) = \text{tuple}(\text{take}_1,\ldots,\text{take}_k, f \circ \text{tuple}(\text{take}_{k+1},\text{take}_{k+2}))$$

(It should be clear that m_k and m'_k are special cases of a more general construct.) We then have

$$\mathcal{D}[\![\text{dcl}]\!] : R \times L \to R \times L \times \underline{S \to S}$$

$$\mathcal{D}[\![\underline{\text{var}}\ \text{ide} := \text{exp}]\!]\ (r,1) = (r[\text{in}_1(1)/\text{ide}],\ 1+1,\ \text{abbr}_1)$$

$$\underline{\text{where}}\ \text{abbr}_1 = \text{set}(1) \sqsupset (\mathcal{E}[\![\text{exp}]\!]\ r\ 1)$$

$$\mathcal{D}[\![\underline{\text{fun}}\ \text{ide}(\text{ide}_1,\ldots,\text{ide}_k);\ \text{exp}]\!]\ (r,1) = (\text{abbr}_1,\ 1,\ [\lambda s.s])$$

$$\underline{\text{where}}\ \text{abbr}_1 = r[\text{in}_2(Y(\lambda g:L \to \underline{E \times \ldots \times E \times S \to E \times S}.\lambda l:L.\text{abbr}_2))/\text{ide}]$$

$$\underline{\text{where}}\ \text{abbr}_2 = \mathcal{E}[\![\text{exp}]\!]\ \text{abbr}_3\ (1+k) \sqsupset \text{set}(1) \sqsupset \ldots \sqsupset m'_{k-1}(\text{set}(1+k-1))$$

$$\underline{\text{where}}\ \text{abbr}_3 = r[\text{in}_2(g)/\text{ide}][\text{in}_1(1)/\text{ide}_1]\ldots[\text{in}_1(1+k-1)/\text{ide}_k]$$

$$\mathcal{D}[\![\underline{\text{proc}}\ \text{ide}(\text{ide}_1,\ldots,\text{ide}_k);\ \text{cmd}]\!]\ (r,1) = (\text{abbr}_1,\ 1,[\lambda s.s])$$

$$\underline{\text{where}}\ \text{abbr}_1 = r[\text{in}_3(Y(\lambda g:(L \to \underline{S \to S}) \to (L \to \underline{E \times \ldots \times E \times S \to S}).$$

$$\lambda c:L \to \underline{S \to S}.\lambda l:L.\ \text{abbr}_2\ ))/\text{ide}]$$

$$\underline{\text{where}}\ \text{abbr}_2 = \mathcal{C}[\![\text{cmd}]\!]\ \text{abbr}_3\ c\ (1+k) \sqsupset \text{set}(1) \sqsupset \ldots \sqsupset m'_{k-1}(\text{set}(1+k-1))$$

$$\underline{\text{where}}\ \text{abbr}_3 = r[\text{in}_3(g)/\text{ide}][\text{in}_1(1)/\text{ide}_1]\ldots[\text{in}_1(1+k-1)/\text{ide}_k]$$

$$\mathcal{D}[\![\text{dcl}_1;\text{dcl}_2]\!](r,1) = (\text{abbr}_1\downarrow 1,\ \text{abbr}_1\downarrow 2,\ \text{abbr}_1\downarrow 3 \sqsupset \text{abbr}_2)$$

$$\underline{\text{where}}\ \text{abbr}_1 = \mathcal{D}[\![\text{dcl}_2]\!](\mathcal{D}[\![\text{dcl}_1]\!](r,1)\downarrow 1,\ \mathcal{D}[\![\text{dcl}_1]\!](r,1)\downarrow 2)$$

$$\underline{\text{where}}\ \text{abbr}_2 = \mathcal{D}[\![\text{dcl}_1]\!]\ (r,1)\downarrow 3$$

The clauses rely on set to create a new location if the requested location does not already exist.

The use of continuation style for commands and direct style for declarations is not essential. For expressions problems arise if a continuation style semantics is wanted. In the discussion below the presence of L will be ignored. The usual equation

$$\begin{aligned}\mathcal{E}[\![\mathrm{exp}_1 \text{ ope } \mathrm{exp}_2]\!]\, r\ k = \quad & \mathcal{E}[\![\mathrm{exp}_1]\!]\, r\ (\lambda v_1. \\ & \mathcal{E}[\![\mathrm{exp}_2]\!]\, r\ (\lambda v_2. \\ & k([\mathrm{operator}](v_1, v_2)\ )))\end{aligned}$$

is not appropriate because the two levels are mixed in a prohibited way. The idea with m_1 might suggest

$$\begin{aligned}\mathcal{E}[\![\mathrm{exp}_1 \text{ ope } \mathrm{exp}_2]\!]\, r\ k = \quad & \mathcal{E}[\![\mathrm{exp}_1]\!]\, r\ (\\ & M_1(\mathcal{E}[\![\mathrm{exp}_2]\!]\, r)(\\ & k \circ [\mathrm{operator}]))\end{aligned}$$

but it is a problem to find M_1. If a store semantics /MiSt76/ had been used it would be feasible to give a continuation style semantics (as in /Nie82/) as well as a direct style semantics. In a sense the use of m_k may be viewed as "implementing" a store semantics locally.

Remark In section 2.1 it was suggested that the type ft possibly might be thought of as code. Even if one accepts this view the above semantics is not a compiler because of the top-level domains participating in the definition of functions and procedures. In /MiSt76/ a compiler is developed by transforming a standard semantics (through a store semantics) to a stack semantics. It would be interesting to investigate whether this process can be formalised as "moving top-level notions out of the definitions of functions and procedures". ///

The second example is a small applicative language of recursion equation schemes that will be studied in chapter 5. The syntax of expressions exp and programs pro is given by

$$\begin{aligned} \text{exp} ::= {} & x_i && (1 \leq i \leq k) \\ | \; & F_i(\text{exp}_1, \ldots, \text{exp}_k) && (1 \leq i \leq n) \\ | \; & \Lambda_i(\text{exp}_1, \ldots, \text{exp}_k) && (1 \leq i) \end{aligned}$$

$$\begin{aligned} \text{pro} ::= {} & \underline{\text{let}}\ F_1(x_1, \ldots, x_k) = \text{exp}_1\ \&\ \ldots \\ & \&\ F_n(x_1, \ldots, x_k) = \text{exp}_n\ \underline{\text{in}}\ \text{exp}_0 \end{aligned}$$

Here k and n are natural numbers that will remain constant throughout. We shall not specify their values except assume that they are greater than 1.

The language consists of auxiliary functions (the A_i), functions defined by mutual recursion (the F_i) and variables (the x_i). The parameter mechanism is call-by-name rather than call-by-value as before. The meaning of a program is a function whose arguments are values for the free variables (in exp_0 above). We shall assume that there is only one data type $\underline{E}$ as this will slightly simplify the notation in chapter 5. The meaning of a program therefore will be an element of

$$F = \underline{E} \times \ldots \times \underline{E \to E}$$

where there are k $\underline{E}$'s to the left of $\underline{\to}$. For environments we may use

$$R = F \times \ldots \times F$$

where there are n F's.

The semantics then is

$$\mathcal{P}[\![\text{pro}]\!] : F$$

$$\mathcal{E}[\![\text{exp}]\!] : R \to F$$

defined by

$$\mathcal{P}[\![\underline{\text{let}} \ldots \underline{\text{in}}\ \text{exp}_0]\!] = \mathcal{E}[\![\text{exp}]\!](Y(\lambda \text{env}:R.\ \text{abbr}_1))$$

$$\underline{\text{where}}\ \text{abbr}_1 = (\mathcal{E}[\![\text{exp}_1]\!]\ \text{env}, \ldots, \mathcal{E}[\![\text{exp}_k]\!]\ \text{env})$$

and

$$\mathcal{E}[\![x_i]\!]\ \text{env} = \text{take}_i$$

$$\mathcal{E}[\![F_i(\text{exp}_1,\ldots,\text{exp}_k)]\!]\ \text{env} =$$

$$(\text{env}\downarrow i) \,\square\, \text{tuple}(\mathcal{E}[\![\text{exp}_1]\!]\ \text{env}, \ldots, \mathcal{E}[\![\text{exp}_k]\!]\ \text{env})$$

$$\mathcal{E}[\![Ai(\text{exp}_1,\ldots,\text{exp}_k)]\!]\ \text{env} =$$

$$a_i \,\square\, \text{tuple}(\mathcal{E}[\![\text{exp}_1]\!]\ \text{env}, \ldots, \mathcal{E}[\![\text{exp}_k]\!]\ \text{env})$$

Here a_i are to be constants of contravariantly pure type (F in fact).

There is no explicit conditional above but it may be assumed to be one of the A_i because the parameter mechanism is call-by-name. It could be introduced explicitly by

$$\mathcal{E}[\![\underline{\text{if}}\ \text{exp}_1\ \underline{\text{then}}\ \text{exp}_2\ \underline{\text{else}}\ \text{exp}_3]\!]\ \text{env} =$$

$$\text{cond}([\lambda e.\ e \text{ represents tt}] \,\square\, \mathcal{E}[\![\text{exp}_1]\!]\ \text{env},$$

$$\mathcal{E}[\![\text{exp}_2]\!]\ \text{env}, \mathcal{E}[\![\text{exp}_3]\!]\ \text{env})$$

but we shall not do so in chapter 5.

3 COLLECTING SEMANTICS

In the previous chapter a metalanguage was specified and its standard interpretation was given. In the next chapter we consider abstract interpretation, i.e. specification of (forward) data flow analyses. As with previous developments of abstract interpretation it is helpful to consider the most precise of all data flow analyses: the "static semantics" of /CoCo77b/ which is called the collecting semantics in this thesis.

The idea is that a function $f: N_\perp \times N_\perp \to N_\perp \times N_\perp$ becomes $g: \mathcal{P}(N_\perp \times N_\perp) \to \mathcal{P}(N_\perp \times N_\perp)$. Here $\mathcal{P}(N_\perp \times N_\perp)$ is a powerdomain which is an analogue within domain theory of the powersets used in the introduction. In section 3.1 several notions of powerdomains are exemplified and the theory is developed for the notion chosen. Section 3.2 studies how to obtain e.g. $\mathcal{P}(N_\perp \times N_\perp)$ from $\mathcal{P}(N_\perp)$ as is necessary in order to define the collecting semantics by merely giving a new interpretation (the collecting interpretation). This motivates a study of the tensor product $\otimes$ because $\mathcal{P}(N_\perp \times N_\perp)$ "is" $\mathcal{P}(N_\perp) \otimes \mathcal{P}(N_\perp)$. Further use of the tensor product will be made in chapter 4. The collecting interpretation is then defined in section 3.3 and a suitable relationship between the standard and collecting semantics is proved. The relationship essentially says that $g(Y) = \{f(y) \mid y \in Y\}$ which means that g is the extension of f to powerdomains.

3.1 RELATIONAL POWERDOMAINS

Powerdomains were developed in order to make it possible to give semantics to nondeterministic and parallel programs. The semantics of a nondeterministic program like

x:=x+1; (x:=x+27 or x:=x+3)

is a function $f:N_\perp \rightarrow \mathcal{P}(N_\perp)$. We define $\mathcal{P}(N_\perp)$ below for three notions of powerdomain. It should be stressed that the general definitions are more complex as will be apparent when the theory of relational powerdomains is covered.

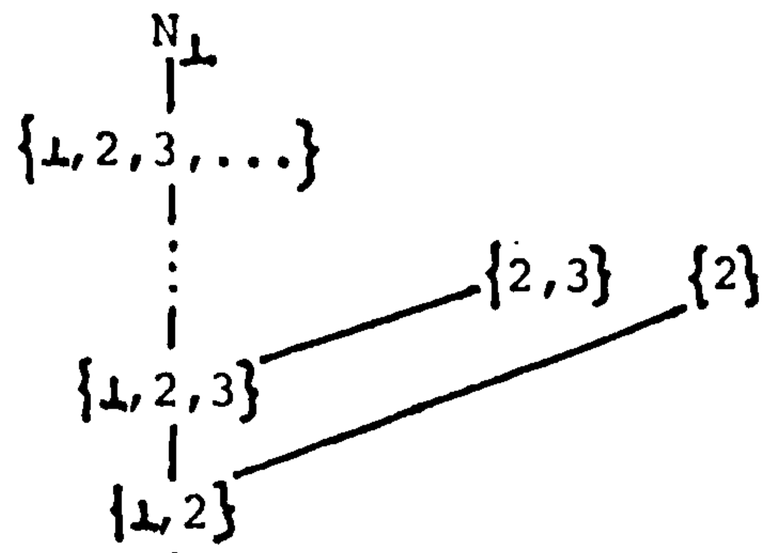
The Plotkin powerdomain $\mathcal{P}_P(N_\perp)$ /Plo76/ has as elements some of the subsets of $N_\perp$. A subset of $N_\perp$ is an element of the powerdomain iff it contains $\perp$ or is finite but nonempty. The partial order is the Egli-Milner order defined by

$X \sqsubseteq_{EM} Y$ iff for all $x \in X$ there is $y \in Y$ such that $x \sqsubseteq y$, and

for all $y \in Y$ there is $x \in X$ such that $x \sqsubseteq y$

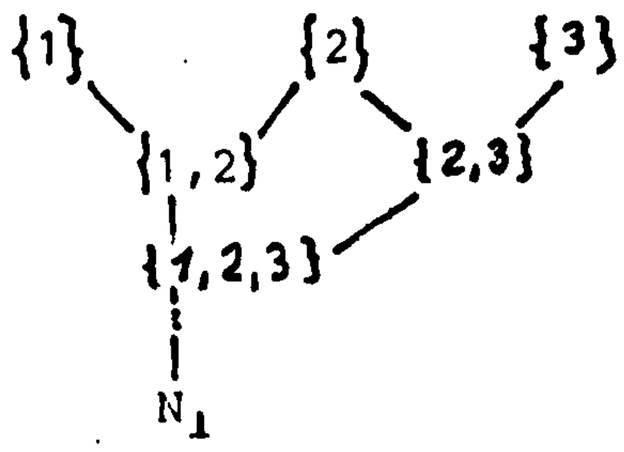
Then $\mathcal{P}_P(N_\perp)$ is a cpo and may be pictured as:

$\mathcal{P}_P(N_\perp)$:



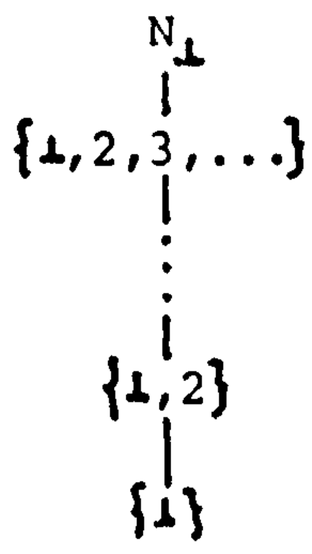
The Smyth powerdomain /Smy78/ has as elements $N_\perp$ and all finite and nonempty subsets of N. The partial order is superset inclusion, i.e.e $X \sqsubseteq Y$ iff $X \supseteq Y$. This may be pictured as:

$\mathcal{P}_S(N_\perp)$:



The relational powerdomain $\mathcal{P}_R(N_\perp)$ (taken from /PloLN/) has as elements all subsets of $N_\perp$ that contain $\perp$ and is partially ordered by subset inclusion, i.e. $X \sqsubseteq Y$ iff $X \subseteq Y$. This may be pictured as:

$\mathcal{P}_R(N_\perp)$:



One may note that $\mathcal{P}_R(N_\perp)$ is isomorphic to the ordinary powerset of N.

It will now be argued that the relational powerdomain is the better choice of the three possibilities. One argument is the isomorphism (in $\underline{\underline{\underline{\text{CPO}}}}$) that was mentioned above and therefore a development based on this powerdomain may be claimed to be "a generalisation" of the usuall theory (e.g. /CoCo77b/). The Smyth powerdomain does not seem uusable because it includes only one infinite subset of $N_\perp$ and in data flow analysis one often considers representations of infinite sets (as in the "detection of signs" example in the introduction).

The formulation of abstract interpretation given in the introduction focuses upon what set of states will be possible at some point (say at the end of the program) not whether it can be guaranteed that the point is always reached. This agrees with the relational powerdomain where all elements contain $\perp$. The Plotkin powerdomain (as was used in /Myc81/) enables one to express that a certain point is always

reached (i.e. that the set does not contain $\perp$). This is useful when extending abstract interpretation to include aspects of termination and is the subject matter of chapter 5. It will be argued, however, that the absence of infinite sets not containing $\perp$ severely limits the additional generality that is obtained (and the major application in /Myc81/ is incorrect). For this reason, and less importantly because the theory of relational powerdomains is simpler, we shall adopt the relational powerdomain.

Definition of the relational powerdomain

To define the relational powerdomain we need the notion of an ideal. A tuple $B = (B,\perp,\sqsubseteq)$ is a pointed quasi order iff $\sqsubseteq$ is a quasi order on B (i.e. a partial order except that $x\sqsubseteq y\sqsubseteq x$ does not imply $x=y$) and $\perp$ is a least element. For $D = (D,\sqsubseteq)$ an algebraic cpo the tuple $(B_D,\perp,\sqsubseteq)$ is a countable pointed quasi order (because B_D is countable).

An ideal I of a pointed quasi order B is a subset of B that contains $\perp$ and is left-closed: if $b\sqsubseteq b'$ and $b'\in I$ then $b\in I$. It is convenient to define $LC_B(X) = \{b\in B \mid \exists x\in X: b\sqsubseteq x\}$ and the index B is usually omitted.

The ideals of $N_\perp$ are exactly the elements of $\wp_R(N_\perp)$. This motivates (omitting the subscript R):

Definition For a cpo D define $\wp(D) = (\{\text{ideals of } B_D\},\subseteq)$. ///

Theorem 3.1:1 /PloLN/. If D is an algebraic cpo then $\wp(D)$ is an algebraic complete lattice with $\bigcup\chi = \{\perp_D\}\cup\bigcup\chi$ and finite elements those ideals I that equal LC(Y) for some finite subset Y of B_D. ///

Proof It is straightforward that $\wp(D)$ is a complete lattice with least upper bounds as stated. If $Y\subseteq B_D$ is finite and nonempty then I=LC(Y)

is a finite element: for if $I \subseteq \bigcup_n I_n$ for some chain $(I_n)_n$ then $Y \subseteq \bigcup_n I_n$ so $Y \subseteq I_n$ for some n and hence $I \subseteq I_n$. We next use fact 2.2:1 so let I be an ideal. Because B_D is countable there is a one-one map $i: B_D \to N$. Then $(LC(\{\perp\} \cup \{b \in B_D \mid b \in I \wedge i(b) \leq n\}))_n$ is a chain of finite elements with least upper bound I. ///

The notion of singletons for powersets motivates defining $\{d\}_R = \{b \in B_D \mid b \sqsubseteq d\} = LC(\{d\})$ whenever $d \in D$. It is straightforward to show that $\lambda d.\{d\}_R$ is a continuous function from D to $\wp(D)$.

Theorem 3.1:2 /PloLN/. Let D be an algebraic cpo and L a complete lattice. For a continuous function $f: D \to L$ there is precisely one continuous and additive function $f^\dagger: \wp(D) \to L$ such that $f^\dagger(\{d\}_R) = f(d)$ for all $d \in D$. It is given by $f^\dagger(I) = \bigsqcup\{f(b) \mid b \in I\}$ and $\lambda f.f^\dagger$ is an isomorphism from $D \to L$ to $\wp(D) \xrightarrow{a} L$ (the cpo of additive as well as continuous functions) with inverse $\lambda g.\lambda d.g(\{d\}_R)$. ///

Proof The explicitly defined $f^\dagger$ is continuous as is shown by

$$f^\dagger(\bigcup_n I_n) = \bigsqcup\{f(b) \mid b \in \bigcup_n I_n\} = \bigsqcup\{\bigsqcup\{f(b) \mid b \in I_n\} \mid n \in N\} = \bigsqcup_n f^\dagger(I_n)$$

and a similar calculation shows that $f^\dagger$ is additive. Next

$$f^\dagger(\{d\}_R) = \bigsqcup\{f(b) \mid b \in B_D \wedge b \sqsubseteq d\} = f(d)$$

because f is continuous and $\{d\}_R$ is a countable and directed subset of D. For any other g satisfying the conditions we show $g(I) = f^\dagger(I)$ for I an ideal of B_D. By g continuous it suffices to consider ideals that are finite elements and by g additive it suffices to consider $I = \{b\}_R$ for $b \in B_D$. The result is then immediate. It follows that $\lambda f.f^\dagger$ is a bijection as stated and it and its inverse are easily seen to be monotonic so that $\lambda f.f^\dagger$ is an isomorphism (in $\underline{\underline{CPO}}$). ///

The above result is useful for defining $\mathcal{P}(...)$ as a functor. Let $\underline{\underline{\underline{ALG}}}$ be the category with algebraic cpo's as objects and continuous functions as morphisms. It is an admissible subcategory of $\underline{\underline{\underline{CPO}}}$ (see the proof for $\underline{\underline{\underline{ACC}}}$ in section 2.2). For a morphism $f:D\to D'$ we define $\mathcal{P}(f) = (\lambda d.\{f(d)\}_R)^\dagger$ and the definition on objects has already been given. This specifies a locally continuous and covariant functor over $\underline{\underline{\underline{ALG}}}$ as is now proved. That $\mathcal{P}(f\cdot g) = \mathcal{P}(f)\cdot\mathcal{P}(g)$ is by the "precisely one" result of the theorem: for $\mathcal{P}(f)\cdot\mathcal{P}(g)$ is continuous and additive and $(\mathcal{P}(f)\cdot\mathcal{P}(g))\{d\}_R = \mathcal{P}(f)\{g(d)\}_R = \{(f\cdot g)(d)\}_R$. One shows $\mathcal{P}(id) = id$ in a similar way. That $\mathcal{P}$ is continuous from $D\to D'$ to $\mathcal{P}(D)\to\mathcal{P}(D')$ is because $\lambda f.f^\dagger$ is continuous from $D\to\mathcal{P}(D')$ to $\mathcal{P}(D)\to\mathcal{P}(D')$; the latter claim follows from the theorem because additivity is an admissible predicate. The theorem states that $\mathcal{P}(D)$ is a complete lattice and it is immediate that $\mathcal{P}(f)$ is strict iff f is. Hence $\mathcal{P}(...)$ is a locally continuous and covariant functor over all of $\underline{\underline{\underline{ALG}}}$, $\underline{\underline{\underline{\underline{ALGs}}}}$, $\underline{\underline{\underline{ACC}}}$, $\underline{\underline{\underline{\underline{ACCs}}}}$.

There are other isomorphic ways of defining the relational powerdomain. One that may have more "intuitive appeal" is the following where members of $\mathcal{P}(D)$ do not only contain elements of B_D. A subset X of D is closed iff it equals $\{d\in D \mid LC(\{d\}) \subseteq LC(X)\}$. Then $\mathcal{P}(D)$ is isomorphic to $(\{\text{nonempty closed subsets of D}\},\subseteq)$ /PloLN/ but this result will not be used subsequently.

Other characterisations

The collecting interpretation uses powerdomains but the general treatment of abstract interpretation in chapter 4 will also consider other algebraic complete lattices. This means we will use the

category $\underline{\underline{\underline{ACL}}}$ that has algebraic complete lattices as objects and continuous functions as morphisms. It is an admissible subcategory of $\underline{\underline{\underline{CPO}}}$ (as follows from an easy modification of the proof for $\underline{\underline{\underline{ACC}}}$). In the remainder of this section we consider what special there is about powerdomains, i.e. we identify the image in $\underline{\underline{\underline{ACL}}}$ of $\mathcal{P}$ acting on $\underline{\underline{\underline{ALG}}}$.

For this we need to define the ideal completion of a countable and pointed quasi order $B = (B,\perp,\sqsubseteq)$. A directed ideal of B is an ideal of B that is a directed subset.

Definition For a pointed quasi order B define

$\bar{B} = (\{\text{directed ideals of } B\},\subseteq)$. ///

Theorem 3.1:3 /PloLN/. If B is a countable and pointed quasi order then $\bar{B}$ is an algebraic cpo. The finite elements of $\bar{B}$ are those of the form $LC(\{b\})$ for $b\in B$. ///

Proof Clearly $\bar{B}$ is a partially ordered set with $\{\perp\}$ as least element. It is a cpo because for $(J_n)_n$ a chain of directed ideals $\bigcup_n J_n$ is a directed ideal and therefore the least upper bound. Clearly an element $LC(\{b\})$ for $b\in B$ is finite. We now once more use fact 2.2:1. Let $i:N\rightarrow B$ be onto and J a directed ideal. Set $b_0=\perp$ and $b_{n+1}=b_n$ if $i(n+1)\notin J$ and otherwise b_{n+1} is chosen as some upper bound in J of b_n and $i(n+1)$. (This uses the countable axiom of choice.) Clearly $(LC(\{b_n\}))_n$ is a chain of finite elements with J as the least upper bound. ///

Fact 3.1:4. If D is an algebraic cpo then D is isomorphic to $\overline{B_D}$. The isomorphism from D to $\overline{B_D}$ is $\lambda d.LC(\{d\})$ and its inverse is $\lambda J.\bigsqcup J$. ///

The characterisation of powerdomains builds on the notion of a prime element. (The definition of prime and irreducible below is dual to that of /GHKLMS80/.) Let $D = (D,\sqsubseteq)$ be an algebraic cpo.

Definition An element $d \in D$ is a prime iff $d \sqsubseteq d_1 \sqcup d_2$ implies that $d \sqsubseteq d_1$ or $d \sqsubseteq d_2$ and it is irreducible iff $d = d_1 \sqcup d_2$ implies that $d = d_1$ or $d = d_2$. ///

Write $PB_D = \{b \in B_D \mid b \text{ is a prime}\}$ and $IB_D = \{b \in B_D \mid b \text{ is irreducible}\}$. If D is consistently complete as well as algebraic and if d is finite then d_1 and d_2 may be assumed finite in the above definition without changing the concept.

Theorem 3.1:5. An algebraic complete lattice L is isomorphic to a powerdomain (of an algebraic cpo) iff

$$B_L = \{b_1 \sqcup \ldots \sqcup b_n \mid n > 0 \wedge b_i \in PB_L\}$$

and then $L \cong \overline{(PB_L)}$. ///

Proof To show "only if" we may without loss of generality assume $L = \wp(D)$ for D an algebraic cpo. The finite elements of L are by 3.1:1 those of the form $\bigcup_{i \leq n} LC(\{b_i\})$ for $n > 0$ and $b_i \in B_D$. It therefore suffices to show that the finite primes are those of the form $LC(\{b\})$ for $b \in B_D$. Clearly such an element is prime and if a finite element $d = \bigcup_{i < n} LC(\{b_i\})$ is not of this form then there is $j < k$ such that b_j and b_k are maximal in $\{b_1, \ldots, b_n\}$ but neither $b_j \sqsubseteq b_k$ nor $b_k \sqsubseteq b_j$. Then using $d_1 = LC(\{b_i \mid i \leq n \wedge b_i \neq b_j\})$ and $d_2 = LC(\{b_i \mid i \leq n \wedge b_i \neq b_k\})$ in the definition of prime shows that d is not prime.

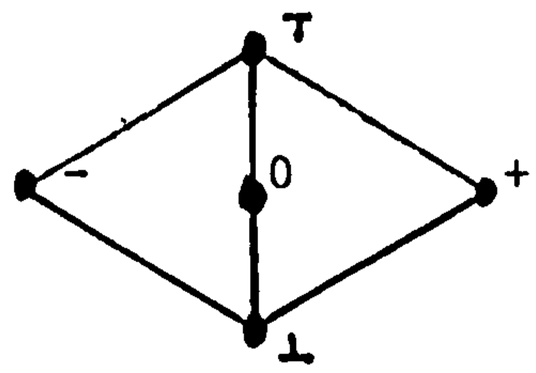
For "if" let L be an algebraic complete lattice satisfying the condition. Then $D = \overline{PB_L}$ is an algebraic cpo and we will show $L \cong \wp(D)$.

Since $L \cong \overline{B_L}$ and B_D is isomorphic to PB_L (in a category with partially ordered sets and continuous functions) it suffices to show that $L' = (\{\text{directed ideals of } B_L\}, \subseteq)$ is isomorphic to $L'' = (\{\text{ideals of } PB_L\}, \subseteq)$. Define $\theta: L' \to L''$ by $\theta(J) = J \cap PB_L$ and $\theta^{-1}: L'' \to L'$ by $\theta^{-1}(I) = \{b_1 \sqcup \ldots \sqcup b_n \mid n>0 \wedge b_i \in I\}$. Note that $b_1 \sqcup \ldots \sqcup b_n$ exists in L (because L is a complete lattice) and gives an element of B_L because each b_i is a finite element of L. So $\theta^{-1}(I)$ is a directed ideal of B_L and $\theta(J)$ is an ideal of PB_L. It is straightforward to show that $\theta(\theta^{-1}(I)) = I$ and $\theta^{-1}(\theta(J)) = J$ using the connection between B_L and PB_L. Finally, both θ and θ^{-1} are monotonic. ///

Further information about powerdomains is given by:

Fact 3.1:6. A prime of an object of $\underline{\underline{ACL}}$ is irreducible but not necessarily conversely. ///

For the proof note that the complete lattice V defined by



has $PB_V = \{\bot\}$, $IB_V = \{-, 0, +, \bot\}$ and $B_V = V$.

Theorem 3.1:7. If D is an algebraic cpo and I is an element of $\wp(D)$ then it is a prime iff it is irreducible iff it is a singleton ($I = \{d\}_R$ for some $d \in D$). ///

Proof If $d \in D$ then $\{d\}_R$ is a prime and hence irreducible: for let $d = \bigsqcup_n b_n$ where $(b_n)_n$ is a chain of finite elements and suppose that $\{d\}_R \subseteq I_1 \cup I_2$. Then without loss of generality I_1 contains b_n for infinitely many n, hence I_1 contains all b_n and $\{d\}_R \subseteq I_1$.

Next we show that an irreducible ideal I must be a singleton. If I is directed then $I=\{\bigcup I\}_R$ so we will show that I must be directed. For a contradiction assume that I is not directed but is irreducible. There are $b_1, b_2 \in I$ that have no upper bound in I. Define the ideals $I_1 = LC(\{b \in I \mid b \not\sqsupseteq b_1\})$ and $I_2 = LC(\{b \in I \mid \neg(b \sqsupseteq b_1)\})$. Then I_1 contains b_1 but not b_2 and I_2 contains b_2 but not b_1. But $I = I_1 \vee I_2$ is immediate and since $I \neq I_1$ and $I \neq I_2$ the desired contradiction has been established. ///

In chapter 4 we shall consider algebraic complete lattices L satisfying $B_L = \{b_1 \sqcup \ldots \sqcup b_n \mid n>0 \wedge b_i \in IB_L\}$. The previous two theorems show that the powerdomains are included (as is V incidentally). The condition of the previous theorem cannot be used to characterise powerdomains:

Fact 3.1:8. The powerdomains are a proper subset of the set of $\underline{\underline{ACL}}$ objects where primes and irreducible means the same. ///

For the proof note that a non-powerdomain satisfying the condition is the following: the elements are $\{\perp\} \cup \{0,1\}^*$ and $u \sqsubseteq v$ iff $u=\perp$ or u is a finite string over $\{0,1\}$ with v as a prefix.

Turning to functions we have:

Theorem 3.1:9. Let D and D' be algebraic cpo's and $g:\wp(D)\to\wp(D')$ a continuous function. There exists a continuous function $f:D\to D'$ such that $g=\wp(f)$ iff g is additive and prime-preserving (i.e. that f(d) is a prime whenever d is). ///

Proof By a previous theorem the primes are the singletons and this shows that $\wp(f)$ is prime-preserving and it is clearly additive. Next given g define $f(d) = \bigsqcup g(\{d\}_R)$ as a continuous function. Using 3.1:2 it follows that $g=\wp(f)$. ///

3.2 TENSOR PRODUCTS

Consider a type $\underline{A}_1\underline{\#}\ldots\underline{\#}\underline{A}_k$ where # is one of the symbols $\times$, $*$, $+$ or $\perp$. In the standard semantics the corresponding domain is $A_1\#\ldots\#A_k$ and in the collecting semantics it will be $\mathcal{P}(A_1\#\ldots\#A_k)$. The problem is how to build $\mathcal{P}(A_1\#\ldots\#A_k)$ in a structural way. This entails finding an operator $\textcircled{\#}$ such that $\mathcal{P}(A_1\#\ldots\#A_k)$ is at least isomorphic to $\mathcal{P}(A_1)\textcircled{\#}\ldots\textcircled{\#}\mathcal{P}(A_k)$. The operator should preferably work for all algebraic complete lattices rather than just powerdomains in order to facilitate the development of abstract interpretation in chapter 4. When # is $\perp$ one may use $\perp$ for $\textcircled{\perp}$ and when # is $+$ one may use $\times$ for $\oplus$. It is less clear what to do when # is $\times$ or $*$ and this motivates a study of tensor products /ArMa75/. A further "pay off" of this is given in chapter 4 where it is claimed that the tensor product generalises the "relational method" described in chapter 1.

DEFINITION AND CONSTRUCTION

Before giving the categorical definition of the tensor product it is convenient to state some auxiliary results. The categories to be considered in this section are $\underline{\underline{\underline{ACL}}}$, $\underline{\underline{\underline{\underline{ACLs}}}}$, $\underline{\underline{\underline{\underline{ACLa}}}}$ and $\underline{\underline{\underline{\underline{\underline{ACLas}}}}}$ where the a means that morphisms are additionally additive.

Lemma 3.2:1. $\underline{\underline{\underline{ACL}}}$, $\underline{\underline{\underline{\underline{ACLs}}}}$, $\underline{\underline{\underline{\underline{ACLa}}}}$ and $\underline{\underline{\underline{\underline{\underline{ACLas}}}}}$ are admissible subcategories of $\underline{\underline{\underline{CPO}}}$. ///

Proof For $\underline{\underline{\underline{ACL}}}$ and $\underline{\underline{\underline{\underline{ACLs}}}}$ this is much as for $\underline{\underline{\underline{ACC}}}$ in section 2.2. To get the rest of the result consider a chain $((D_n)_n,(e_n)_n)$ in $\underline{\underline{\underline{\underline{\underline{ACLae}}}}}$ (or equally $\underline{\underline{\underline{\underline{\underline{\underline{ACLase}}}}}}$) and let $(D,(r_n)_n)$ be the limiting cone in $\underline{\underline{\underline{\underline{CPOe}}}}$. To see that each r_n is a morphism of $\underline{\underline{\underline{\underline{\underline{ACLae}}}}}$ it suffices to show that

each r_n^U is additive. But each e_n^U is additive and therefore D as constructed in 2.2:2 has componentwise binary least upper bounds. It then follows from the formula for r_n^U that the function is additive. ///

Define an (upper) semi-lattice as a partially ordered set $L = (L,\underline{\sqsubseteq})$ where all binary least upper bounds exist.

...

Lemma 3.2:2. An algebraic cpo is in $\underline{\underline{ACL}}$ iff it is a semi-lattice.///

Proof Only "if" is non-trivial so let D be an algebraic cpo that is also a semi-lattice. Let X be a subset of D and note that $\bigsqcup X$ exists if X is finite. For a countable set $X = \{x_n \mid n \geq 0\}$ we know that $\bigsqcup_n \bigsqcup\{x_1,\ldots,x_n\}$ exists and it is easy to see this is the least upper bound of X. For a general set X we know that $\bigsqcup(LC(X))$ exists and it is easy to see this is $\bigsqcup X$. ///

In much the same way it follows that:

Lemma 3.2:3. A morphism of $\underline{\underline{ACL}}$ is in $\underline{\underline{ACLas}}$ iff it is completely additive and is in $\underline{\underline{ACLa}}$ iff it preserves least upper bounds of all non-empty families. ///

The notion of seperate additivity is central to the definition of the tensor product. A function $f:L_1\times\ldots\times L_k\rightarrow L$ (for $k\geq 2$) is seperately additive (respectively seperately continuous, seperately strict) iff for each i and $(l_1,\ldots,l_k)$ the function $\lambda l.f(l_1,\ldots,l,\ldots,l_k)$ is additive (respectively continuous, strict). Seperate strictness implies strictness but not conversely whereas seperate additivity is implied by additivity but not conversely. It follows from section 2.2 that seperate continuity is the same as continuity.

The definition of the tensor product is concerned with transforming a seperately additive function to an additive function.

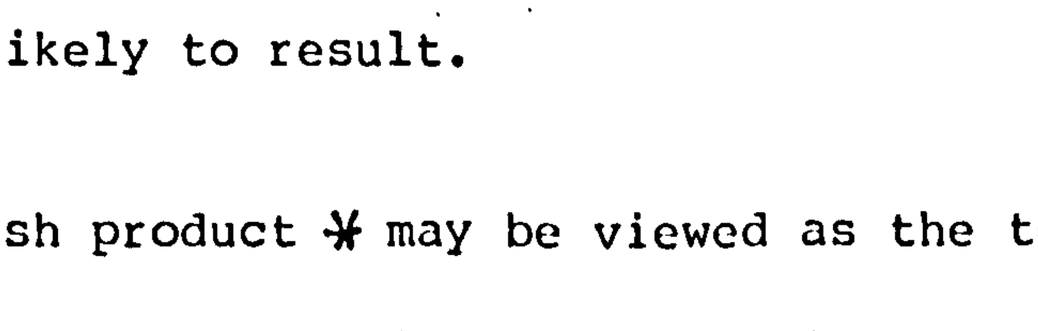
Definition A tensor product in $\underline{\underline{\underline{ACL}}}$ for $\times$ with respect to additivity assigns to any k objects $L_1,\ldots,L_k$ an object $L_1\otimes\ldots\otimes L_k$ and a seperately additive morphism

$$\text{cross}_\times : L_1 \times \ldots \times L_k \to L_1 \otimes \ldots \otimes L_k$$

that is called the inclusion. Furthermore, for every seperately additive morphism $f:L_1\times\ldots\times L_k\to L$ of $\underline{\underline{\underline{ACL}}}$ there is precisely one additive morphism $f^\times:L_1\otimes\ldots\otimes L_k\to L$ such that $f^\times\cdot\text{cross}_\times=f$. The morphism $f^\times$ is called the extension of f. ///

It should be clear that similar definitions may be given by using other entities for $\underline{\underline{\underline{ACL}}}$, $\times$ and additivity. In this way one may define a tensor product $\circledast$ in $\underline{\underline{\underline{\underline{ACLs}}}}$ for $*$ with respect to additivity. The inclusion is written cross_* and the extension of f as f^*. In this section the theory will be developed for both $\otimes$ and $\circledast$ but the presentation will focus upon $\otimes$. The index $\times$ to cross will be omitted when confusion is not likely to result.

Example The smash product $*$ may be viewed as the tensor product in $\underline{\underline{\underline{CPO}}}$ of $\times$ with respect to strictness. The inclusion is smash and the extension of f is the function f restricted to the domain $L_1*\ldots*L_k$. This amounts to saying that for every continuous and seperately strict $f:L_1\times\ldots\times L_k\to L$ there is precisely one strict and continuous function $f^\times:L_1*\ldots*L_k\to L$ such that



commutes. ///

The two major questions to be addressed are whether a tensor product $\otimes$ exists and if more than one tensor product exists then what is their relationship. The second question is answered by:

Fact 3.2:4. Let $\otimes$ be a tensor product with inclusion $cross_{\chi}$ and extension f^{χ}. Similarly $\otimes'$, $cross_{\chi'}$, and $f^{\chi'}$. Then for all $L_1, \ldots, L_k$ there is an isomorphism

$$\theta: L_1 \otimes \ldots \otimes L_k \rightarrow L_1 \otimes' \ldots \otimes' L_k$$

such that $\theta \cdot cross_{\chi} = cross_{\chi'}$ and $f^{\chi} \cdot \theta^{-1} = f^{\chi'}$. It is given by $\theta = (cross_{\chi'})^{\chi}$ and $\theta^{-1} = (cross_{\chi})^{\chi'}$. ///

The proof is similar to the one showing that two limiting cones are isomorphic.

Construction

To show that a tensor product always exists we give a particular construction that uses the algebraicity of objects. The idea /PloPS/ is first to construct a certain logic, then obtain a countable and pointed quasi order from it and finally define the tensor product as the ideal completion.

Let $L_1, \ldots, L_k$ be $\underline{\underline{ACL}}$ objects and note that all of $B_{L_1}, \ldots, B_{L_k}$ are semi-lattices with a least element. The logic T_{χ} is now constructed using ordinary logical notions /Men63/. It is convenient to formulate it as a k+1 sorted logic, where the sorts are $\underline{L}_1, \ldots, \underline{L}_k$ and $\underline{L}_0$ for the tensor product. The constant terms are b_i (or $b_i : \underline{L}_i$) of sort $\underline{L}_i$ whenever b_i is a finite element of L_i. The function symbols

are $\otimes:\underline{L}_1\times\ldots\times\underline{L}_k\to\underline{L}_0$ and $\sqcup:\underline{L}_0\times\underline{L}_0\to\underline{L}_0$. For formulae there is only one predicate = that takes two terms of sort $\underline{L}_0$ as arguments. These are subject to the following

axioms 1. $\vdash x = x$

2. $\vdash x \sqcup x = x$

3. $\vdash x \sqcup y = y \sqcup x$

4. $\vdash x \sqcup (y \sqcup z) = (x \sqcup y) \sqcup z$

5. $\vdash \otimes(b_1,\ldots,b_i,\ldots,b_k) \sqcup \otimes(b_1,\ldots,b_i',\ldots,b_k)$
$= \otimes(b_1,\ldots,b_i \sqcup b_i',\ldots,b_k)$

inference rules

1. $$\frac{\vdash x = y}{\vdash y = x}$$

2. $$\frac{\vdash x = y \qquad \vdash y = z}{\vdash x = z}$$

3. $$\frac{\vdash x = y}{\vdash x \sqcup z = y \sqcup z}$$

Axiom 1 and the inference rules express that = is a congruence. Axioms 2, 3 and 4 give properties that hold for a binary least upper bound operator. Axiom 5 expresses that $\otimes$ is seperately additive. The convention is used that $b_i \sqcup b_i'$ is the constant term b_i'' it equals rather than assuming $\sqcup$ is a function $\underline{L}_i\times\underline{L}_i\to\underline{L}_i$. Generally t will denote a term of sort $\underline{L}_0$ and $\vdash t_1=t_2, \ldots, \vdash t_{n-1}=t_n$ may be abbreviated to $\vdash t_1=\ldots=t_n$. The logic T$*$ is constructed similarly but with function symbol $\circledast$ instead of $\otimes$ and with the additional

axiom 6. $\vdash \circledast(b_1,\ldots,\perp,\ldots,b_k) = \circledast(\perp,\ldots,\perp)$

Proofs in one logic can sometimes be transformed to proofs in another. Suppose that $T\chi'$ is constructed similarly to $T\chi$ but from objects $L'_1, \ldots, L'_k$.

Lemma 3.2:5. If $r_i : B_{L_i} \to B_{L'_i}$ are additive functions then $\vdash_{T\chi} t_1 = t_2$ implies $\vdash_{T\chi'} t_1|^r = t_2|^r$ where $|^r$ is defined by $(t_1 \sqcup t_2)|^r = t_1|^r \sqcup t_2|^r$ and $\otimes(b_1,\ldots,b_k)|^r = \otimes(r_1(b_1),\ldots,r_k(b_k))$. ///

The similar result for $T\not\chi$ also needs r_i to be strict. The proof of the lemma is formally by induction on the proof $\vdash t_1 = t_2$. Informally one must show that the axioms of $T\chi$ still hold in $T\chi'$ when transformed by $|^r$ and that the transformed inference rules are derived rules in $T\chi'$. We omit the details.

The countable and pointed quasi order $P\chi$ ($P\not\chi$) is now constructed from the logic $T\chi$ ($T\not\chi$). The countable set of elements is the set of terms of sort L_0 and $\otimes(\perp,\ldots,\perp)$ ($\not\otimes(\perp,\ldots,\perp)$) will be the least element. The quasi order is defined by $t_1 \sqsubseteq t_2$ iff $\vdash t_1 \sqcup t_2 = t_2$. That this gives a quasi order is straightforward using axioms 2 and 4. It is not a partial order as two syntactically different terms may well be proved equal. Because of axiom 4 we may omit parentheses around terms so any term is of the form $\otimes(b_1^1,\ldots,b_k^1) \sqcup \ldots \sqcup \otimes(b_1^n,\ldots,b_k^n)$. That the designated element is least now follows from the following two results.

Lemma 3.2:6. The term $t_1 \sqcup t_2$ is a least upper bound of terms t_1 and t_2 with respect to $\sqsubseteq$. ///

Proof That $t_2 \sqsubseteq t_1 \sqcup t_2$ is by $\vdash t_2 \sqcup (t_1 \sqcup t_2) = t_2 \sqcup t_2 \sqcup t_1 = t_1 \sqcup t_2$ and $t_1 \sqsubseteq t_1 \sqcup t_2$ is similar. If $t_1 \sqsubseteq t$ and $t_2 \sqsubseteq t$ then $t_1 \sqcup t_2 \sqsubseteq t$ is because

$\vdash t_1 \sqcup t_2 \sqcup t = t_1 \sqcup t = t.$ ///

Lemma 3.2:7. $\otimes(b_1,\ldots,b_k) \sqsubseteq \otimes(b'_1,\ldots,b'_k)$ iff $b_i \sqsubseteq b'_i$ for all i (or, in the case of P$\ast$, if some b_j is $\perp$). ///

Proof For "if" it suffices to illustrate the case k=2:

$$\vdash \otimes(b_1,b_2) \sqcup \otimes(b'_1,b'_2) = \otimes(b_1,b_2) \sqcup \otimes(b_1,b'_2) \sqcup \otimes(b'_1,b'_2)$$
$$= \otimes(b_1,b'_2) \sqcup \otimes(b'_1,b'_2) = \otimes(b'_1,b'_2)$$

using axiom 5 several times. For "only if" we use a soundness result: a model specifies a set B (for L_0) and functions $\otimes'$ and $\sqcup'$ and the predicate ='. Consider the model where B is B_{L_i}, $\otimes'$ is $\downarrow i$, $\sqcup'$ is the binary least upper bound operation of L_i and =' is equality. Since $\downarrow i$ is (seperately) additive all axioms are true and the inference rules preserve truth. Hence (by induction on the proof of) $\vdash t_1 = t_2$ implies $[\![t_1]\!] = [\![t_2]\!]$, where $[\![\ ]\!]$ gives the value of a term in this model. So $\otimes(b_1,\ldots,b_k) \sqsubseteq \otimes(b'_1,\ldots,b'_k)$ gives

$$\vdash \otimes(b_1,\ldots,b_k) \sqcup \otimes(b'_1,\ldots,b'_k) = \otimes(b'_1,\ldots,b'_k)$$

so $b_i \sqcup b'_i = b'_i$ and then $b_i \sqsubseteq b'_i$ as was to be shown. In the case of P$\ast$ we use $\circledast'(b_1,\ldots,b_k)$ that is b_i if all $b_j \neq \perp$ and $\perp$ otherwise. ///

The "tensor product" $L_1 \otimes \ldots \otimes L_k$ is now defined to be $\overline{P\times}$ (where $\overline{\ldots}$ is the completion by ideals of section 3.1) and similarly $L_1 \circledast \ldots \circledast L_k$ is $\overline{P+}$. It follows from section 3.1 that these definitions give algebraic cpo's. That they are objects of $\underline{\underline{\underline{\underline{ACL}}}}$ ($\underline{\underline{\underline{\underline{ACL}}}}_{\perp}$) follows from lemmas 3.2:2, 3.2:6 and

Lemma 3.2:8. If a countable and pointed quasi order has least upper bounds of pairs then its completion by ideals is a semi-lattice.///

Proof Define $J_1 \cup J_2 = \{b \mid \exists b_1 \in J_1, b_2 \in J_2 : b \sqsubseteq b_1 \cup b_2\}$ and note that this gives a directed ideal that is an upper bound of J_1 and J_2. Furthermore it is contained in any other upper bound that is also a directed ideal. ///

In a similar way one shows that the least upper bound operation in $L_1 \otimes \ldots \otimes L_k$ is

$$\bigsqcup \chi = \{\otimes(\perp, \ldots, \perp)\} \cup \{t \mid \exists n>0 : t_1, \ldots, t_n \in U\chi : t \sqsubseteq t_1 \cup \ldots \cup t_n\}$$

and similarly for $L_1 \circledast \ldots \circledast L_k$.

To verify that the above construction gives a tensor product we must define the inclusion and extension of functions. Define

$$\text{cross}_{\times}((l_1, \ldots, l_k)) =$$
$$\{t \mid \exists b_1 \in B_{L_1}, \ldots, b_k \in B_{L_k} : (\forall i : b_i \sqsubseteq l_i) \wedge t \sqsubseteq \otimes(b_1, \ldots, b_k)\}$$

and similarly for $\circledast$. It is straightforward to verify that $\text{cross}_{\times}$ and cross_{*} are strict and continuous functions that produce directed ideals.

Lemma 3.2:9. $\text{cross}_{\times}$ and cross_{*} are seperately additive. ///

Proof Let $l', l'' \in L_i$ and $b \in B_{L_i}$ be such that $b \sqsubseteq l' \cup l''$. Then there exists $b', b'' \in B_{L_i}$ such that $b \sqsubseteq b' \cup b''$ and $b' \sqsubseteq l'$ and $b'' \sqsubseteq l''$. The proof is then straightforward using the formula for least upper bounds in $L_1 \otimes \ldots \otimes L_k$. Similarly for $\circledast$. ///

Given an $\underline{\underline{\underline{\text{ACL}}}}$ morphism $f: L_1 \times \ldots \times L_k \to L$ define $f^{\times}$ by

$$f^{\times}(J) = \bigsqcup\{f(b_1, \ldots, b_k) \mid \otimes(b_1, \ldots, b_k) \in J\}$$

This defines a continuous function that satisfies $f^{\times}(\text{cross}(l_1, \ldots, l_k)) = f(l_1, \ldots, l_k)$ because by continuity it suffices to consider finite l_i and for these the result is immediate. For an

$\underline{\underline{\text{ACLs}}}$ morphism $f:L_1*\ldots*L_k \to L$ define f^* by

$$f^*(J) = \bigsqcup\{f(\text{smash}(b_1,\ldots,b_k)) \mid \circledast(b_1,\ldots,b_k)\in J\}$$

This gives a strict and continuous function such that $f^*\cdot\text{cross}_\times = f$.

Lemma 3.2:10. $f^\times$ is additive if f is seperately additive and similarly for f^*. ///

Proof We calculate

$$f^\times(J'\sqcup J'') =$$

$$\bigsqcup\{f(b_1,\ldots,b_k) \mid \otimes(b_1,\ldots,b_k) \sqsubseteq \otimes(b'_1,\ldots,b'_k) \sqcup \otimes(b''_1,\ldots,b''_k)$$

$$\wedge\ \otimes(b'_1,\ldots,b'_k)\in J' \wedge \otimes(b''_1,\ldots,b''_k)\in J''\} =$$

(see below)

$$\bigsqcup\{f(b_1,\ldots,b_k) \mid \otimes(b_1,\ldots,b_k)\in J' \vee \otimes(b_1,\ldots,b_k)\in J''\} =$$

$$f^\times(J') \sqcup f^\times(J'')$$

For the middle equality it is immediate that $\sqsupseteq$ holds and for $\sqsubseteq$ we shall use a soundness result as in the proof of 3.2:7. The model has B to be B_L, $\otimes'$ to be f, $\sqcup'$ to be binary least upper bound in L and $='$ is equality. It then follows that

$$\otimes(b_1,\ldots,b_k) \sqsubseteq \otimes(b'_1,\ldots,b'_k) \sqcup \otimes(b''_1,\ldots,b''_k)$$

implies

$$f(b_1,\ldots,b_k) \sqsubseteq f(b'_1,\ldots,b'_k) \sqcup f(b''_1,\ldots,b''_k)$$

For $\circledast$ use $\circledast'$ that is f·smash. ///

The correctness of the construction now follows:

Theorem 3.2:11. The constructed $L_1\otimes\ldots\otimes L_k$ is a tensor product with inclusion $\text{cross}_\times$ defined above and extension of f given by $f^\times$ above. Similarly for $\otimes$. ///

Proof It remains to show that $f^{\times}$ is the unique continuous and additive function h such that h•cross=f. So let h be some other candidate and show $h(J)=f^{\times}(J)$. By continuity it suffices to consider arguments J of the form $\{t \mid t \Subset t'\}$ and by additivity one may assume $t'=\otimes(b'_1,\ldots,b'_k)$. For such arguments the result follows from h•cross = $f^{\times}$•cross. ///

The function $\lambda f.f^{\times}$ is continuous because it is an isomorphism from the continuous and seperately additive functions $L_1\times\ldots\times L_k \to L$ to the continuouos and additive functions $L_1\otimes\ldots\otimes L_k \to L$. Similarly $\lambda f.f^{*}$ is an isomorphism from strict, continuous and seperately additive functions to completely additive functions.

Extending it to a functor

All domain constructors have been defined as functors and we now consider how to do this for tensor products. For $\underline{\underline{\underline{\text{ACL}}}}$ morphisms $f_i:L_i \to M_i$ define the $\underline{\underline{\underline{\text{ACL}}}}$ morphism

$$f_1\otimes\ldots\otimes f_k = (\text{cross}_{\times}\text{•}f_1\times\ldots\times f_k)^{\times}$$

This gives an additive (respectively strict) morphism if all the f_i are additive (respectively strict) because of 3.2:10 and $\text{cross}^{\bullet}(f_1\times\ldots\times f_k)$ then is seperately additive. Similarly for $\underline{\underline{\underline{\underline{\text{ACLs}}}}}$ morphisms $f_i:L_i \to M_i$ define the $\underline{\underline{\underline{\underline{\text{ACLs}}}}}$ morphism

$$f_1\circledast\ldots\circledast f_k = (\text{cross}_{*}\text{•}f_1 * \ldots * f_k)^{*}$$

Again this gives an additive morphism if all the f_i are. Both $\otimes$ and $\circledast$ are locally continuous and covariant functors over $\underline{\underline{\underline{\underline{\underline{\text{ACLas}}}}}}$. This will be a consequence of a later result but may easily be proved directly using the following idea: since $\times$ (and $*$) are functors one may use 3.2:11 to show the functor laws in much the same way the result was shown for $\wp$ in section 3.1. Neither of $\otimes$ or $\circledast$ gives a functor over

ACLs, however, because the lack of additivity means that the law for composition may fail as is shown by the following example.

Example Let $L = \wp(\{0,1,2\}_\perp)$, $M = \wp(\{-1,0,1\}_\perp)$ and recall V defined in section 3.1. Define $h:L \to V$ and $g:V \to M$ as the naming of the elements suggests, e.g. $h(\{\perp\}) = \perp$, $h(\{1,2,\perp\}) = +$ etc.. Consider $J = cross(\{1\}_R, \{0\}_R) \cup cross(\{2\}_R, \{1\}_R)$ in $L \otimes L$. Then

$$(h \otimes h)(J) = cross(+,0) \cup cross(+,+) = cross(+,+)$$

so that

$$((g \otimes g)^\bullet (h \otimes h))(J) = cross(\{1\}_R, \{-1,0,1,\perp\})$$

This differs from

$$((g^\bullet h) \otimes (g^\bullet h))(J) = cross(\{1\}_R, \{0\}_R) \cup cross(\{1\}_R, \{1\}_R)$$
$$= cross(\{1\}_R, \{0,1,\perp\})$$

Note that g is not additive (but h is). ///

For the purposes of chapter 4 it is convenient not to restrict consideration to ACLas. We therefore define a weaker notion than covariant functor and show that this includes $\otimes$ and $\circledast$. For morphisms in a sub cpo-category of CPO it has already been defined what additivity means. This may be extended to morphisms in a product category by defining such a morphism to be additive iff all its components are.

Definition Consider two cpo-categories with a notion of additivity defined as above. A (upper) semi-functor from one to another is a map upon objects and a map upon morphisms such that

i) $F(id) = id$

ii) $F(f^\bullet f') \sqsubseteq F(f)^\bullet F(f')$

iii) F(f) is additive if f is

iv) equality holds in ii) when f is additive ///

This definition includes semi-functors of more than one argument. The functors $\otimes$, $\oplus$ and $\perp$ are semi-functors over $\underline{\underline{\text{ACLs}}}$ as follows from Fact 3.2:12. Every covariant functor (over a cpo-category) that preserves additivity is a semi-functor. ///

Fact 3.2:13. Every semi-functor gives rise to a covariant functor on the subcategory of additionally additive morphisms. ///

Theorem 3.2:14. $\otimes$ is a locally continuous semi-functor over $\underline{\underline{\text{ACL}}}$ and $\underline{\underline{\text{ACLs}}}$ and $\circledast$ is over $\underline{\underline{\text{ACLs}}}$. ///

Proof We only consider $\otimes$ and the only non-trivial result is that $(f_1 \otimes \ldots \otimes f_k) \cdot (f'_1 \otimes \ldots \otimes f'_k) \sqsupseteq (f_1 \cdot f'_1) \otimes \ldots \otimes (f_k \cdot f'_k)$ and that equality holds when all f_i are additive. Abbreviating the inequality to $g \cdot g' \sqsupseteq g''$ we have

$$\begin{aligned}(g \cdot g')(J) &= g(\bigsqcup\{\text{cross}(f'_1(b_1),\ldots,f'_k(b_k)) \mid \otimes(b_1,\ldots,b_k) \in J\}) \\ &\sqsupseteq \bigsqcup\{g(\text{cross}(f'_1(b_1),\ldots,f'_k(b_k)) \mid \otimes(b_1,\ldots,b_k) \in J\} \\ &= \bigsqcup\{\text{cross}(f_1(f'_1(b_1)),\ldots,f_k(f'_k(b_k)) \mid \otimes(b_1,\ldots,b_k) \in J\} \\ &= g''(J)\end{aligned}$$

If all f_i are additive then so is g. Since g is continuous and all objects are algebraic we have

$$g(\bigsqcup \chi) = \bigsqcup\{g(J) \mid J \in \chi\}$$

whenever χ is not empty. This shows that equality holds in the above calculation when all f_i are additive. ///

It is straightforward to show that the composition of locally continuous semi-functors gives a locally continuous semi-functor and that

the tupling of locally continuous semi-functors gives a locally continuous semi-functor.

OTHER CHARACTERISATIONS

In the remainder of this section we shall study the tensor product of powerdomains, view the tensor product as a subcpo of a powerdomain, view the tensor product as a function space and study commutativity and associativity of $\otimes$ and $\circledast$.

Powerdomains

The motivation for studying the tensor product was to achieve that $\wp(A_1 \times \ldots \times A_k)$ is isomorphic to $\wp(A_1) \otimes \ldots \otimes \wp(A_k)$. That this is the case follows from fact 3.2:4 and

Theorem 3.2:15. $\wp(D_1 \times \ldots \times D_k)$ with inclusion $\text{cross}_\times = \lambda(I_1,\ldots,I_k).I_1 \times \ldots \times I_k$ is a tensor product of $\wp(D_1)$, ..., $\wp D_k)$ with respect to $\times$ in $\underline{\underline{\underline{\text{ACL}}}}$ and the extension of f is given by

$$f^\times = \lambda I.\ \bigcup\{f(\{b_1\}_R,\ldots,\{b_k\}_R) \mid (b_1,\ldots,b_k) \in I\} \qquad ///$$

Proof Clearly cross is continuous and seperately additive and $f^\times$ is continuous and additive and satisfies $f^\times \cdot \text{cross} = f$ (because this is immediate for finite arguments). Any other candidate for $f^\times$ equals $f^\times$ on arguments of the form $\{(b_1,\ldots,b_k)\}_R$ and by additivity on all finite elements and by continuity always. ///

A similar result holds for $*$. Here $\wp(D_1 * \ldots * D_k)$ is a tensor product (of $*$ in $\underline{\underline{\underline{\underline{\text{ACLs}}}}}$) with inclusion

$$\text{cross}_* = \lambda(I_1,\ldots,I_k).\ \{\text{smash}(b_1,\ldots,b_k) \mid b_i \in I_i\}$$

and the extension f^* of f is defined similarly to $f^\times$.

The isomorphism between $\rho(A_1 \times \ldots \times A_k)$ and $\rho(A_1) \otimes \ldots \otimes \rho(A_k)$ can be strengthened to include morphisms.

Definition /ArMa75/. A natural transformation from a functor $F: \underline{\underline{L}} \to \underline{\underline{K}}$ to a functor $G: \underline{\underline{L}} \to \underline{\underline{K}}$ is a family nat of $\underline{\underline{K}}$ morphisms indexed by the objects of $\underline{\underline{L}}$ such that whenever $f: L \to L'$ is a morphism of $\underline{\underline{L}}$ the equality $G(f) \cdot nat(L) = nat(L') \cdot F(f)$ holds. This means that

$$\begin{array}{ccc} F(L) & \xrightarrow{nat(L)} & G(L) \\ \downarrow{\scriptstyle F(f)} & & \downarrow{\scriptstyle G(f)} \\ F(L') & \xrightarrow{nat(L')} & G(L') \end{array}$$

commutes. A natural equivalence is a natural transformation nat where each nat(L) is an isomorphism. ///

Theorem 3.2:16. There is a natural equivalence from $\rho^{\cdot}\times : \underline{\underline{ALG}}^k \to \underline{\underline{ACL}}$ to $\otimes^{\cdot}(\rho^{\cdot}P_1, \ldots, \rho^{\cdot}P_k)$. Similarly for $\rho^{\cdot}*$ and $\otimes^{\cdot}(\rho^{\cdot}P_1, \ldots, \rho^{\cdot}P_k)$. ///

Proof Let cross and $f^\times$ refer to the tensor product $\rho(D_1 \times \ldots \times D_k)$ and cross' and $f^{\times'}$ refer to $\rho(D_1) \otimes \ldots \otimes \rho(D_k)$. Define $nat(D_1, \ldots, D_k) = (cross')^\times$ and note that fact 3.2:4 shows that this is an isomorphism. It is straightforward to show that

$$\rho(f_1 \times \ldots \times f_k)(I) = (cross \cdot \rho(f_1) \times \ldots \times \rho(f_k))^\times (I)$$

holds for morphisms $f_i: D_i \to D_i'$. It then follows that $(cross')^\times \cdot \rho(f_1 \times \ldots \times f_k) \cdot cross$ equals $\rho(f_1) \otimes \ldots \otimes \rho(f_k) \cdot (cross')^\times \cdot cross$ and by the tensor product property of $\rho(D_1 \times \ldots \times D_k)$ this shows the result. ///

The tensor product as a subcpo of a powerdomain

We next study the quasi order in $P_{\times}$ and $P_{\not\times}$ with the aim of giving a different characterisation of it and thereby of the tensor product. The basic idea is to consider a function rep that associates a term t with the tuples $(b_1,\ldots,b_k)$ such that $\otimes(b_1,\ldots,b_k) \sqsubseteq t$. The definition will be more "explicit", however, and the above description will follow from the next theorem. Define

$$\text{init}(t) = \text{LC}(\{(b_1,\ldots,b_k) \mid \otimes(b_1,\ldots,b_k) \text{ mentioned in } t\})$$

and note that this gives an ideal of $B_{L_1} \times \ldots \times B_{L_k}$ hence an element of $\wp(L_1 \times \ldots \times L_k)$. Next define

$$\text{step}(I) = \bigcup_{i=1}^{k} \text{step}_i(I)$$

$$\text{step}_i(I) = \{(b_1,\ldots,b_k) \mid \exists b_i', b_i'' : b_i \sqsubseteq b_i' \sqcup b_i'' \wedge (b_1,\ldots,b_i',\ldots,b_k) \in I \wedge (b_1,\ldots,b_i'',\ldots,b_k) \in I\}$$

$$\text{close}(I) = \text{LFP}(\lambda I'.\ \text{step}(I \cup I'))$$

$$= \bigcup_{n \in N} \text{step}^n(I)$$

This defines continuous and extensive (e.g. $\text{step}(I) \supseteq I$) functions over $\wp(L_1 \times \ldots \times L_k)$ and close is the least fixed point of step that contains I. Finally define

$$\text{rep}(t) = \text{close}(\text{init}(t))$$

For $\circledast$ the change is to replace $(b_1,\ldots,b_k)$ in init and step_i by $\text{smash}(b_1,\ldots,b_k)$ so as to operate on $\wp(L_1 * \ldots * L_k)$.

Theorem 3.2:17. $\vdash t_1 = t_2$ iff $\text{rep}(t_1) = \text{rep}(t_2)$ ///

Proof "Only if" is by induction on the proof of $\vdash t_1 = t_2$. The result is immediate for axioms 1, 2, 3, 4, (6) and for axiom 5 it suffices to observe that $\text{init}(t_1) \subseteq \text{init}(t_2)$ and $\text{step}(\text{init}(t_1)) \supseteq \text{init}(t_2)$.

It is straightforward to verify the result for inference rules 1 and 2 and for inference rule 3 note that

$$\text{rep}(t \cup t') =$$

$$\text{close}(\text{init}(t) \vee \text{init}(t')) =$$

$$\text{close}(\text{rep}(t) \vee \text{rep}(t'))$$

For "if" it suffices to show that $\vdash \otimes(b_1,\ldots,b_k) \sqcup t_2 = t_2$ holds for every $\otimes(b_1,\ldots,b_k)$ mentioned in t_1. For then $\vdash t_1 \sqcup t_2 = t_2$ and by symmetry $\vdash t_2 \sqcup t_1 = t_1$ so that $\vdash t_1 = t_2$. If $\otimes(b_1,\ldots,b_k)$ is mentioned in t_1 the tuple $(b_1,\ldots,b_k)$ is an element of $\text{rep}(t_1)$ and hence of $\text{step}^n(\text{init}(t_2))$ for some n. It follows that it suffices to prove

$$(b_1,\ldots,b_k) \in \text{step}^n(\text{init}(t_2))$$

$$\text{implies} \vdash \otimes(b_1,\ldots,b_k) \sqcup t_2 = t_2$$

by induction on n. The result is immediate for n=0 so let $(b_1,\ldots,b_k)$ be an element of $\text{step}^{n+1}(\text{init}(t_2))$. This means that there is i, b_i' and b_i'' such that $b_i \sqsubseteq b_i' \sqcup b_i''$ and $(b_1,\ldots,b_i',\ldots,b_k)$ and $(b_1,\ldots,b_i'',\ldots,b_k)$ are both elements of $\text{step}^n(\text{init}(t_2))$. The inductive hypothesis then gives

$$\vdash t_2 = t_2 \sqcup \otimes(b_1,\ldots,b_i',\ldots,b_k) \sqcup \otimes(b_1,\ldots,b_i'',\ldots,b_k)$$

and since

$$\vdash \otimes(b_1,\ldots,b_i',\ldots,b_k) \sqcup \otimes(b_1,\ldots,b_i'',\ldots,b_k)$$

$$= \otimes(b_1,\ldots,b_i,\ldots,b_k) \sqcup \otimes(b_1,\ldots,b_i',\ldots,b_k) \sqcup \otimes(b_1,\ldots,b_i'',\ldots,b_k)$$

it follows that $\vdash t_2 = t_2 \sqcup \otimes(b_1,\ldots,b_i,\ldots,b_k)$ as was to be shown. (For $\circledast$ one must take special care with tuples $(b_1,\ldots,b_k)$ where some but not all components are $\perp$.) ///

Corollary 3.2:18. $t_1 \sqsubseteq t_2$ iff $\text{rep}(t_1) \subseteq \text{rep}(t_2)$. ///

The proof uses that $\text{rep}(t_1 \sqcup t_2) = \text{close}(\text{rep}(t_1) \cup \text{rep}(t_2))$ and that

this equals $rep(t_2)$ iff $rep(t_1) \sqsubseteq rep(t_2)$. For later reference note the formulae

$$rep(\otimes(b_1,\ldots,b_k)) = \{(b_1',\ldots,b_k') \mid \forall i: b_i' \sqsubseteq b_i\}$$

$$rep(\otimes\!\!\!\!\otimes(b_1,\ldots,b_k)) = \{smash(b_1',\ldots,b_k') \mid \forall i: b_i' \sqsubseteq b_i\}$$

This characterisation of the quasi orders in $P\times$ and $P\ast$ is convenient for characterising the irreducible elements and the primes.

Lemma 3.2:19. $\otimes(b_1,\ldots,b_k)$ is irreducible iff all b_i are (or, in the case of $\otimes$, if some b_i is $\perp$). ///

Proof "Only if" is straightforward using 3.2:7 so consider "if" for $P\times$. Let all b_i be irreducible and suppose for the sake of contradiction that $\otimes(b_1,\ldots,b_k)$ is not. Then there are t_1, t_2 such that $\vdash \otimes(b_1,\ldots,b_k) = t_1 \sqcup t_2$ but neither $\vdash \otimes(b_1,\ldots,b_k) = t_1$ nor $\vdash \otimes(b_1,\ldots,b_k) = t_2$. From the previous theorem $(b_1,\ldots,b_k)$ is neither an element of $rep(t_1)$ nor $rep(t_2)$ but is an element of $rep(t_1 \sqcup t_2) = close(rep(t_1) \cup rep(t_2))$. For the contradiction we prove by induction on n that $(b_1,\ldots,b_k)$ is not an element of $step^n(rep(t_1) \cup rep(t_2))$. The base case is immediate so suppose $b_i \sqsubseteq b_i' \sqcup b_i''$ with $(b_1,\ldots,b_i',\ldots,b_k)$ and $(b_1,\ldots,b_i'',\ldots,b_k)$ elements of $step^n(rep(t_1) \cup rep(t_2))$. Since the latter set is a subset of $rep(t_1 \sqcup t_2) = LC(\{(b_1,\ldots,b_k)\})$ it follows that $b_i' \sqsubseteq b_i$, $b_i'' \sqsubseteq b_i$ and hence $b_i = b_i' \sqcup b_i''$. By b_i irreducible this contradicts the inductive hypothesis and therefore the proof of the inductive step is complete. The proof of "if" for $P\ast$ is similar.///

Lemma 3.2:20. $\otimes\!\!\!\!\otimes(b_1,\ldots,b_k)$ is a prime iff all b_i are (or, in the case of $\otimes\!\!\!\!\otimes$, if some b_i is $\perp$). ///

Proof "Only if" is straightforward using 3.2:7. "If" may be proved by adapting the proof of the previous lemma or by a soundness result as

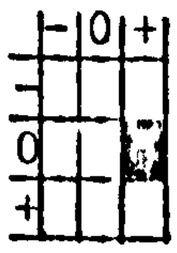
in the proof of 3.2:7: the model has $B=\{\tau\}_\perp$ and $\otimes'(b'_1,\ldots,b'_k)=\tau$ iff all $b'_i \sqsupseteq b_i$ (or, in the case of $\otimes$, if some b_i is $\perp$). That axiom 5 holds is because the b_i are primes. ///

From this lemma it immediately follows that a tensor product of powerdomains is (isomorphic to) a powerdomain. For by theorem 3.1:5 an object L of $\underline{\underline{\underline{\underline{ACL}}}}$ is a powerdomain iff PB_L generates B_L, i.e. $B_L = \{b_1 \cup \ldots \cup b_n \mid n>0 \wedge b_i \in PB_L\}$, and the lemma shows that the finite elements $\{t \mid t \sqsubseteq t'\}$ of the tensor product fulfils this condition. In section 4.5 it will be of interest to know that lemma 3:2.19 implies that the tensor product preserves the property that IB_L generates B_L.

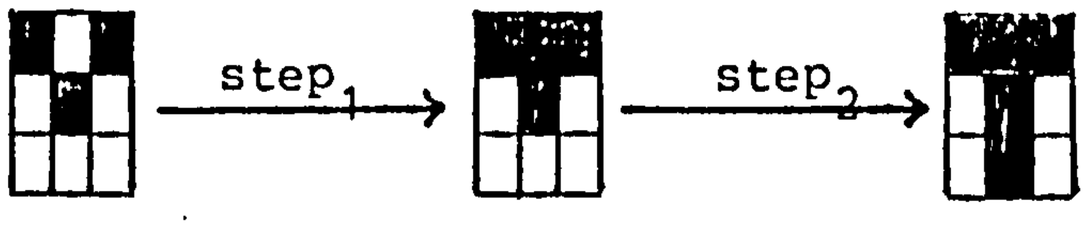
Another application of theorem 3.2:17 is to give a pictorial representation of a tensor product $L_1 \otimes \ldots \otimes L_k$. The method illustrated in the example below works whenever L_i are complete lattices of finite cardinality such that IB_{L_i} is a flat cpo that generates $L_i = B_{L_i}$.

Example The five element complete lattice V of section 3.1 is the smallest complete lattice that is not (isomorphic to) a powerdomain. The set $A_V = IB_V - \{\perp\}$ contains only incomparable elements (i.e. if b and b' are elements then $b \sqsubseteq b'$ implies b=b') and furthermore for every element of V there is a subset of A_V whose least upper bound equals that element.

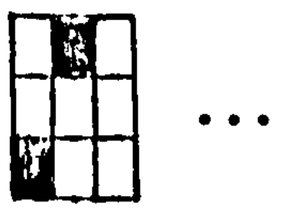
The elements $\{t' \mid t' \sqsubseteq t\}$ of $V \otimes V$ can be uniquely represented by $rep(t) \cap A_V \times A_V$. Such a representation may be depicted by black squares on a 3×3 chess board. As an example $t=\otimes(0,+)$ gives



The partial order on V ⊛ V gives rise to the following ordering of chess boards: $cb_1 \sqsubseteq cb_2$ iff each black square of cb_1 is also black in cb_2. The function step may be defined on such chess boards. This implies that the chess board for $rep(t) \wedge A_V \times A_V$ may be obtained from that for $init(t) \wedge A_V \times A_V$ by repeated application of this function. The analogue of $step_1$ operates upon a row as follows. Let H be the set of elements corresponding to black squares. For every element b in A_V such that b ∈ H the square it corresponds to is painted black. As an example

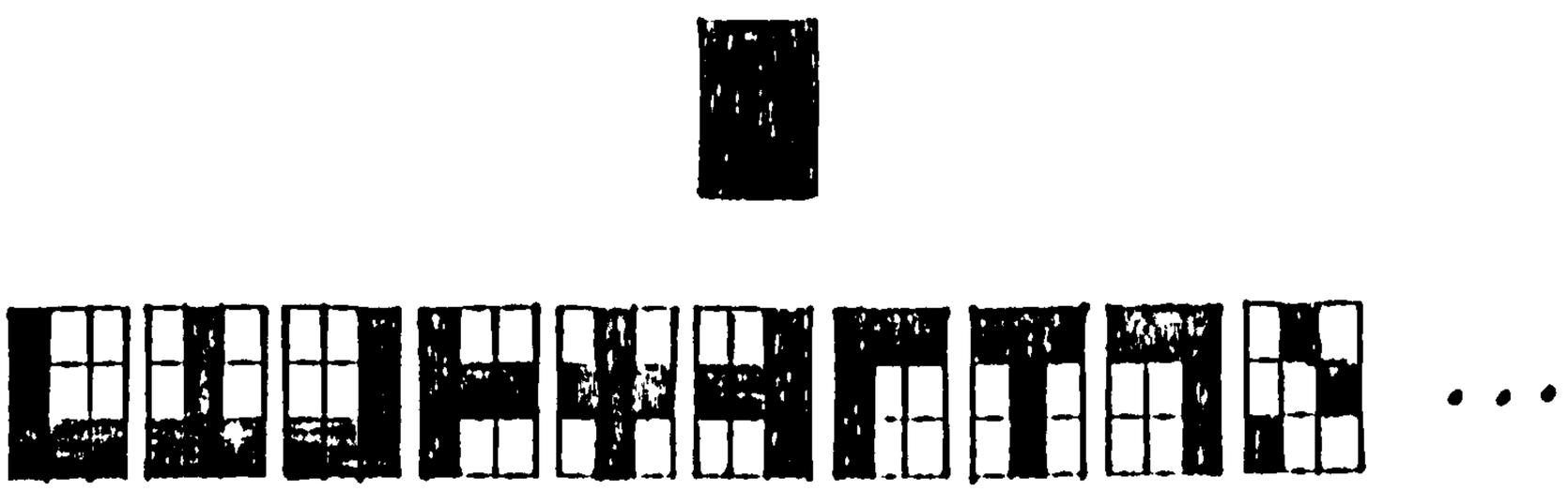


Based on these considerations it is straightforward, but tedious, to prove that V⊛V is isomorphic to the partial order depicted below. Counting from the bottom the elements are divided into levels 0 to 4. A chess board is on level i iff i is the minimal number of black squares so that the chess board can be obtained by repeated application of step to a chess board with that many black squares. The T shaped chess board above therefore is on level 3. On level 2

 ...

abbreviates all solutions to the 2 rook problem on a 3×3 chess board, i.e. all ways of placing two rooks so neither captures the other. Similarly for the 3 rook problem on level 3.

Between two adjacent levels there should be lines that represent the partial order. The partial order satisfies that if x and y are different chess boards such that x ⊑ y then the level of x is strictly less than the level of y.



///

The function rep was defined as a map from the terms of the logic $T_{\mathcal{X}}$ to $\mathcal{P}(L_1 \times \ldots \times L_k)$. We now extend this to show that $L_1 \otimes \ldots \otimes L_k$ is embedded in $\mathcal{P}(L_1 \times \ldots \times L_k)$. A similar development can be performed for $\circledast$ and $\#$ as well. Define

$$Fix_{step} = (\{I \in \mathcal{P}(L_1 \times \ldots \times L_k) \mid step(I) = I\}, \subseteq)$$

and note that by /Tar55/ this is a complete lattice.

Theorem 3.2:21. $L_1 \otimes \ldots \otimes L_k$ is isomorphic to Fix_{step}. ///

Proof We will explicitly construct the isomorphism in both directions. Define $\Theta: L_1 \otimes \ldots \otimes L_k \rightarrow Fix_{step}$ by $\Theta(J) = \bigcup\{rep(t) \mid t \in J\}$ and note that this specifies a monotonic function into $\mathcal{P}(L_1 \times \ldots \times L_k)$. To see this is a fixed point of step consider $(b_1, \ldots, b_k) \in step(\Theta(J))$. There is i and $t', t'' \in J$ and b_i', b_i'' such that $b_i \sqsubseteq b_i' \sqcup b_i''$ and $(b_1, \ldots, b_i', \ldots, b_k) \in rep(t')$ and similarly for b_i'' and t''. By J a directed ideal it contains $t' \sqcup t''$ and $(b_1, \ldots, b_i, \ldots, b_k) \in rep(t' \sqcup t'') \subseteq \Theta(J)$. This shows that

$step(\theta(J)) \subseteq \theta(J)$ and the other inclusion is immediate.

In the other direction $\theta^{-1}: Fix_{step} \rightarrow L_1 \otimes \ldots \otimes L_k$ is defined as a monotone function by

$$\theta^{-1}(I) = \{\otimes(b_1^1,\ldots,b_k^1) \cup \ldots \cup \otimes(b_1^n,\ldots,b_k^n) \mid n>0 \wedge (b_1^i,\ldots,b_k^i) \in I\}$$

This is a directed ideal because I is a fixed point of step. It then remains to show that $\theta \cdot \theta^{-1} = id$ and $\theta^{-1} \cdot \theta = id$. That $\theta(\theta^{-1}(I)) \supseteq I$ is immediate. If $(b_1,\ldots,b_k) \in \theta(\theta^{-1}(I))$ then there is $(b_1^i,\ldots,b_k^i) \in I$ such that

$$(b_1,\ldots,b_k) \in close(\{(b_1^1,\ldots,b_k^1), \ldots, (b_1^n,\ldots,b_k^n)\})$$

Since I is a fixed point of step it follows that $(b_1,\ldots,b_k) \in I$. That $\theta^{-1}(\theta(J)) \supseteq J$ is immediate. If $t \in \theta^{-1}(\theta(J))$ then $init(t) \subseteq \theta(J)$. The set $\{rep(t) \mid t \in J\}$ is directed and since init(t) is the left-closure of a finite set it follows that there is a term $t' \in J$ such that $init(t) \subseteq rep(t')$. This shows that $t \sqsubseteq t'$ and therefore $t \in J$. ///

The isomorphism θ defined in the above proof satisfies that $\otimes(b_1,\ldots,b_k) \in J$ iff $(b_1,\ldots,b_k) \in \theta(J)$. Using this and fact 3.2:4 one can show that Fix_{step} is a tensor product with inclusion

$$cross(l_1,\ldots,l_k) = \{(l_1,\ldots,l_k)\}_R$$

and extension of a seperately additive function f given by

$$f^{\times}(I) = f^{\dagger}(I)$$

The definition as a functor upon additive functions f_i is

$$(f_1 \otimes \ldots \otimes f_k)(I) = close(\cup\{\{(f_1(b_1),\ldots,f_k(b_k))\}_R \mid (b_1,\ldots,b_k) \in I\})$$

This follows because the least upper bound of a nonempty set χ is $\cup\chi$ in $\wp(L_1^{\times} \ldots {}^{\times} L_k)$ and $close(\cup\chi)$ in Fix_{step}. If χ is a directed

set then close may be omitted because $\cup\chi$ is already a fixed point of step. This shows that

$$\Theta : L_1 \otimes \ldots \otimes L_k \to \wp(L_1 \times \ldots \times L_k)$$

is continuous and it follows that Fix_{step} is a sub-cpo of $\wp(L_1 \times \ldots \times L_k)$, i.e. that it has the same least element and the same least upper bounds of chains.

...

Note For the similar results for $\otimes$ one should insert smash the "obvious" places.

The tensor product $\otimes$ as a function space

It is also possible to view the tensor product $\otimes$ as a certain function space. To formulate this define $D^{op} = (D, \sqsupseteq)$ whenever $D = (D, \sqsubseteq)$. Recall that $A \cong B$ means that A and B are isomorphic (as objects of $\underline{\underline{\mathrm{CPO}}}$) and that $\to_{as}$ means the space of completely additive functions.

Theorem 3.2:22. $L_1 \otimes \ldots \otimes L_k \cong (L_1 \to_{as} \ldots \to_{as} L_{k-1} \to_{as} (L_k{}^{op}))^{op}$ ///

Proof We prove the isomorphism in two steps. First note that

$$(L_1 \to_{as} \ldots \to_{as} (L_k{}^{op}))^{op} \cong (B_{L_1} \to_{as} \ldots \to_{as} (L_k{}^{op}))^{op}$$

where $\to_{as}$ on the righthand side means completely additive functions from a semi-lattice to a complete lattice. When k=2 the isomorphism from right to left is

$$\lambda f.\lambda 1.\ \textstyle\prod\{f(b) \mid b \in B_{L_1} \wedge b \sqsupseteq 1\}$$

Throughout this proof all $\sqsubseteq$ and $\prod$ refer to the L_i so $\prod$ is the least upper bound operator on $L_k{}^{op}$.

Define $\Theta: L_1 \otimes \ldots \otimes L_k \to (B_{L_1} \to_{as} \ldots \to_{as} (L_k{}^{op}))^{op}$ by

$$\Theta(J) = \lambda b_1. \ldots \lambda b_{k-1}. \bigsqcup \{b_k \mid \otimes(b_1, \ldots, b_k) \in J\}$$

We shall assume k=2 below as this simplifies the notation but this is not otherwise important. First we must show that $\Theta(J)$ is completely additive. Strictness is immediate as $\Theta(J)(\perp) = \top$ which is the least element of $L_2{}^{op}$. Additivity amounts to showing

$$\Theta(J)(b' \sqcup b'') = \Theta(J)(b') \sqcap \Theta(J)(b'')$$

It is immediate to show $\sqsubseteq$ because $b' \sqsubseteq b''$ and $\otimes(b'', b_2) \in J$ implies that $\otimes(b', b_2) \in J$ so $\Theta(J)(b') \sqsupseteq \Theta(J)(b'')$. For $\sqsupseteq$ the algebraicity of L_k ensures that if suffices to show for $b_k \in B_{L_k}$ that

$$b_k \sqsubseteq \Theta(J)(b') \sqcap \Theta(J)(b'') \text{ implies } b_k \sqsubseteq \Theta(J)(b' \sqcup b'')$$

The set $\{b_k \mid \otimes(b, b_k) \in J\}$ is directed for all choices of finite elements b so for a finite element b_k the condition $b_k \sqsubseteq \Theta(J)(b)$ is equivalent to $\otimes(b, b_k) \in J$. The result then follows because $\otimes(b' \sqcup b'', b_k) \in J$ whenever $\otimes(b', b_k)$ and $\otimes(b'', b_k)$ are elements of J. Finally, Θ is monotonic because $J_1 \subseteq J_2$ implies that $\Theta(J_1)(b) \sqsubseteq \Theta(J_2)(b)$ holds in L_k.

In the other direction define

$$\Theta^{-1}(f) = \{\otimes(b_1^1, \ldots, b_k^1) \sqcup \ldots \sqcup \otimes(b_1^n, \ldots, b_k^n) \mid n > 0 \wedge \forall i \leq n: b_k^i \sqsubseteq f(b_1^i, \ldots, b_{k-1}^i)\}$$

This is a directed ideal: it is clearly nonempty and it is left-closed because $f: B_{L_1} \to_{as} \ldots \to_{as} L_k{}^{op}$ is monotonic. The function Θ^{-1} is monotonic because $\forall b_1: f(b_1) \sqsubseteq f'(b_1)$ implies that $\Theta^{-1}(f) \sqsubseteq \Theta^{-1}(f')$. It remains to verify that Θ is a bijection with inverse Θ^{-1}.

That $\Theta^{-1}(\Theta(J)) \supseteq J$ is straightforward as both sides are directed ideals so it suffices to consider terms of the form $\otimes(b_1, \ldots, b_k)$.

The converse inclusion follows because $\circledast(b_1,\ldots,b_k)$ is an element of J iff $b_k \sqsubseteq \theta(J)(b_1)\ldots(b_{k-1})$ as was remarked above for $k=2$. That $\theta(\theta^{-1}(f)) \sqsupseteq f$ is straightforward using the algebraicity of L_k because if $b_k \sqsubseteq f(b_1)\ldots(b_{k-1})$ then $\circledast(b_1,\ldots,b_k) \in \theta^{-1}(f)$ so that $b_k \sqsubseteq \theta(\theta^{-1}(f))(b_1)\ldots(b_{k-1})$. For the converse relation suppose that $b_k \sqsubseteq \theta(\theta^{-1}(f))(b_1)\ldots(b_{k-1})$ so that $\circledast(b_1,\ldots,b_k) \in \theta^{-1}(f)$ and it follows that $b_k \sqsubseteq f(b_1,\ldots,b_{k-1})$. ///

For this formulation of the tensor product one may again use fact 3.2:4 to show that

$$\text{cross}(l_1,\ldots,l_k) = \lambda l'_1.\ \ldots\ \lambda l'_{k-1}.(\bigwedge_{j<k} l'_j \sqsubseteq l_j) \to l_k, \perp$$

$$f^* = \lambda g.\ \bigsqcup\{f(l_1,\ldots,l_k) \mid l_k \sqsubseteq g(l_1)\ldots(l_{k-1})\}$$

$$f_1 \circledast \ldots \circledast f_k = \lambda g.\lambda l'_1.\ \ldots\ \lambda l'_{k-1}.$$
$$\bigsqcup\{f_k(g(l_1)\ldots(l_{k-1})) \mid \forall j<k:\ l'_j \sqsubseteq f_j(l_j)\}$$

where f is seperately additive and strict, each f_i additive and strict and $\sqsubseteq$ and $\bigsqcup$ refer to the L_i. It follows from the theorem that for a countable set S we have the isomorphism

$$\wp(S_\perp) \circledast L \cong S \to L$$

because both sides are isomorphic to $(S_\perp \to_s L^{op})^{op}$. This in turn might suggest a connection to the reduced cardinal power of /CoCo79/ (see example 10.2.0.2) but it remains to formally investigate the connection.

Note It is not clear how to obtain a "nice" analogue of the previous theorem for the tensor product $\otimes$. In the search the isomorphism $(L \circledast M)_\perp \cong L_\perp \circledast M_\perp$ may possibly be of use.

Remark It turns out that the previous theorem is not novel (see e.g. /GHKLMS80,Ban80/). For another well known result let an adjunction from L to M be a pair of monotonic adjoined functions (as in the introduction) and let adjunctions be ordered as in CPO2s. Then (f,g) is an adjunction from L to M iff f is completely additive and $g(m) = \bigsqcup\{l \mid f(l) \sqsubseteq m\}$ (see e.g. /CoCo79/). This shows that

$$L \otimes M \cong (\text{adjunctions from } L \text{ to } M^{op})^{op}$$

...

and by defining Galois connections as in /CoCo79/ it follows that

$$L \otimes M \cong (\text{Galois connections from } L \text{ to } M)$$

This result is stated in /Joh82/ for products of the so-called locales and is attributed to a 1979 paper by D. Wigner. ///

Commutativity and associativity

Finally, we comment upon the use of k-ary rather than just binary domain constructors. This is mainly a matter of convenience. It is well known that this is so for cartesian product and smash product as e.g. $L_1 \times L_2 \times L_3$ is isomorphic to $(L_3 \times L_1) \times L_2$ and there is a similar close connection between $f_1 \times f_2 \times f_3$ and $(f_3 \times f_1) \times f_2$. Formally there is a natural equivalence from $\times = \times^{\bullet}(P_1, P_2, P_3)$ to $\times^{\bullet}(\times^{\bullet}(P_3, P_1), P_2)$. We shall, somewhat informally, phrase this as saying that $\times$ (as a family $\times^k$ of functors) is commutative and associative. This holds for $\ltimes$ as well.

Theorem 3.2:23. $\otimes$ is commutative and associative (as a family of functors) over ACLa and similarly $\ltimes$ is over ACLas. ///

Proof We concentrate on $\otimes$ and begin with defining some concepts. A list of terms over $L_1, \ldots, L_k$ is a list $gt_1, \ldots, gt_m$ of terms where

each L_i occurs exactly once and $\otimes$ is the only domain constructor used (possibly used with different arities). For k=3 examples include $L_1 \otimes L_2 \otimes L_3$ and $L_3 \otimes L_2$, L_1 but not L_3, $L_2 \otimes L_2$. We shall not distinguish between the term and the object it denotes. Two lists $gt_1,\ldots,gt_m$ and $gt'_1,\ldots,gt'_m$, over $L_1,\ldots,L_k$ are <u>similar</u> iff

- for each object L of $\underline{\underline{\underline{ACL}}}$ there is an isomorphism Θ_L from the seperately additive morphisms $gt_1 \times \ldots \times gt_m \rightarrow L$ to the seperately additive morphisms $gt'_1 \times \ldots \times gt'_m \rightarrow L$,
- the isomorphisms satisfy that $h \cdot \Theta_L(f) = \Theta_{L'}(h \cdot f)$ whenever $h: L \rightarrow L'$ is an additive morphism.

This concept is of interest because $L_1, \ldots, L_k$ and gt are similar iff gt is a tensor product of $L_1,\ldots,L_k$. The proof of "if" is straightforward so consider "only if". The claim is that gt is a tensor product with inclusion cross $= \Theta_{gt}^{-1}(id_{gt})$ and $\Theta_L(f)$ the extension of $f: L_1 \times \ldots \times L_k \rightarrow L$. To see that $f = \Theta_L(f) \cdot$cross it suffices to show that

$$\Theta_L(f) = \Theta_L(f) \cdot \Theta_{gt}(\Theta_{gt}^{-1}(id)) = \Theta_L(\Theta_L(f) \cdot \Theta_{gt}^{-1}(id))$$

and the remaining conditions are straightforward.

The concept of similarity admits the following commutative property. If p is a permutaion of $1,\ldots,m$ then $gt_1,\ldots,gt_m$ and $gt_{p(1)},\ldots,gt_{p(m)}$ are clearly similar lists. For associativity we state the property that $gt_1,\ldots,gt_m$ is similar to $gt_1 \otimes \ldots \otimes gt_n, gt_{n+1}, \ldots, gt_m$. To sketch the proof it is convenient to assume n=2 and m=3 and define

$$\Theta(f) = \lambda(x_{12},x_3).(\lambda(x_1,x_2).\ f(x_1,x_2,x_3))^{\times} x_{12}$$

It is straightforward to check that Θ satisfies the conditions. If

gt_1 and gt_2 are any two "reorderings" of $L_1 \otimes \ldots \otimes L_k$ then repeated application of the above rules show that gt_1 and gt_2 are similar. Hence both gt_1 and gt_2 are tensor product objects.

Each tensor product gt_i gives rise to a functor $G_i : \underline{\underline{ACLa}}^k \rightarrow \underline{\underline{ACLa}}$ such that $G_i(L_1,\ldots,L_k) = gt_i$. Since tensor products are isomorphic there is an isomorphism

$$nat(L_1,\ldots,L_k) : G_1(L_1,\ldots,L_k) \rightarrow G_2(L_1,\ldots,L_k)$$

As in the proof of 3.2:16 it follows that nat is a natural equivalence from G_1 to G_2. ///

The theorem can be extended to apply to $\otimes$ (and $\circledast$) when viewed as semi-functors over $\underline{\underline{ACL}}$ ($\underline{\underline{ACLs}}$) but we shall not go into the proof of this. Also the functions $take_i$ and functional tuple ... can be shown to be related by the natural equivalence. This shows that also for tensor products a binary domain constructor would suffice for the entire development.

3.3 COLLECTING INTERPRETATION

In this section the collecting interpretation $\underline{C}$ is defined and its properties are studied. The objects in the bottom-level are essentially the powerdomains of the corresponding objects in the standard interpretation $\underline{S}$. The functions in the bottom-level are essentially obtained by applying the powerdomain functor to the corresponding functions in $\underline{S}$. We begin with defining the type part of $\underline{C}$ and then formally prove a relationship with the type part of $\underline{S}$. Then the expression part is treated in a similar manner. The relationships are not quite as trivial as the previous overview might

suggest because the equality between objects / functions in $\underline{C}$ and $\wp$ of objects / functions in $\underline{S}$ only hold for base types / constants of contravariantly pure type.

THE TYPE PART

The type part of $\underline{C}$ is defined by specifying the following three components:

- the category $\underline{C}(\underline{\underline{B}})$ is $\underline{\underline{ACL}}$ and $\underline{C}(\underline{\underline{Bp}})$ is $\underline{\underline{ACLas}}$,
- the domain constructors are $\underline{C}(\underline{X})=\otimes$, $\underline{C}(\underline{*})=\circledast$ $\underline{C}(\underline{+})=\times$ and $\underline{C}(\underline{\perp})=\perp$,
- the constant functors are $\underline{C}(\underline{A}_i) = \wp\cdot\underline{S}(\underline{A}_i)$

This definition satisfies the conditions required in section 2.3. In particular $\underline{\underline{ACL}}$ is a sub cpo-category of $\underline{\underline{CPO}}$ and from lemma 3.2:1 it follows that $\underline{\underline{ACLas}}$ is an admissible subcategory of $\underline{\underline{CPOs}}$. It follows from theorem 3.2:14 and fact 3.2:13 that $\otimes$ and $\circledast$ are locally continuous and covariant functors over $\underline{\underline{ACLas}}$ and it is immediate from section 2.2 that also $\times$ and $\perp$ are.

The definition of the constant functors should come as no surprise. The use of $\otimes$ and $\circledast$ was motivated by theorem 3.2:16, namely that $\wp(D\times E)$ essentially is $\wp(D)\otimes\wp(E)$, and the proof of theorem 3.3:4 states similar results that motivate the use of $\times$ and $\perp$. The choice of using $\underline{\underline{ACL}}$ for $\underline{C}(\underline{\underline{B}})$ will be discussed in the next paragraph. Given the choice, however, the use of $\underline{\underline{ACLas}}$ for $\underline{C}(\underline{\underline{Bp}})$ is "forced". For $\otimes$ and $\circledast$ are not functors over $\underline{\underline{ACL}}$ but are when strictness and additivity are assumed.

The category $\underline{\underline{ACL}}$ is not the only possible choice for $\underline{C}(\underline{\underline{B}})$, another being the subcategory that is the image of $\underline{\underline{ACC}}$ under $\wp$. The use of $\underline{\underline{ACL}}$ is motivated by section 4.1, where it will be argued that $\underline{\underline{ACL}}$

is a natural category to use for abstract interpretation and indeed data flow analysis in general. In the introduction it was stated that the collecting semantics should be thought of as the most precise of all approximating interpretations. It is to facilitate this view that $\underline{\underline{\underline{ACL}}}$ is used in the collecting interpretation. It is a consequence of this decision that the tensor product of non power-domains is needed already in this chapter.

....

Remark Another possibility might be to use $\underline{\underline{\underline{\underline{ACL}}}}_a$ for $\underline{C}(\underline{\underline{B}})$ because, as is shown in section 4.1, this is a possible category for a certain class of approximating interpretations. To make this feasible it is essential that top-level objects are not required to be algebraic. This is because there exists a domain M such that the cpo $M \xrightarrow[a]{} M$ of continuous and additive functions is not algebraic.

Take $M = (\{\bot, \top, 0, 1, \ldots\}, \sqsubseteq)$ where $x \sqsubseteq y$ iff $x = \bot$ or $y = \top$ or $x = y$. Then the identity id is neither finite nor the least upper bound of a chain of finite elements. To see that id is not finite consider the chain $(g_n)_n$ defined by

$$g_n(\bot) = \bot$$

$$g_n(\top) = \top$$

$$g_n(m) = (m \leq n) \rightarrow \top, m+1$$

and note that $id \sqsubseteq \bigsqcup_n g_n$ even though each g_n is incomparable with id. Next suppose by way of contradiction that $id = \bigsqcup_n h_n$ for a chain $(h_n)_n$ of finite elements. Then there exists n_0 such that $n \geq n_0$ implies that h_n is the identity on $\bot$, $\top$, 0, 1. Since each h_n is strictly less than id the set $\{m \mid h_n(m) = \bot\}$ must be nonempty and in fact infinite for each n. When $n \geq n_0$ and m and m' are chosen such that $m \neq m'$ and $h_n(m) = \bot = h_n(m')$ this shows that h_n is not additive and the desired contradiction has

been established. ///

To formulate and prove the relationship between the bottom-level domains of S and C it is helpful to state the following notations and facts concerning natural equivalences. Most of the notation makes sense for natural transformations in general.

Fact 3.3:1. A natural equivalence $id[F]$ from the covariant functor F to itself is defined by $id[F](L) = id_{F(L)}$. When F is the identity functor we shall write this simply id. If nat is a natural equivalence from F to G then nat^{-1} defined by $nat^{-1}(L) = nat(L)^{-1}$ is from G to F. If additionally nat' is a natural equivalence from G to H then nat'•nat defined by $(nat' \bullet nat)(L) = nat'(L) \bullet nat(L)$ is from F to H.///

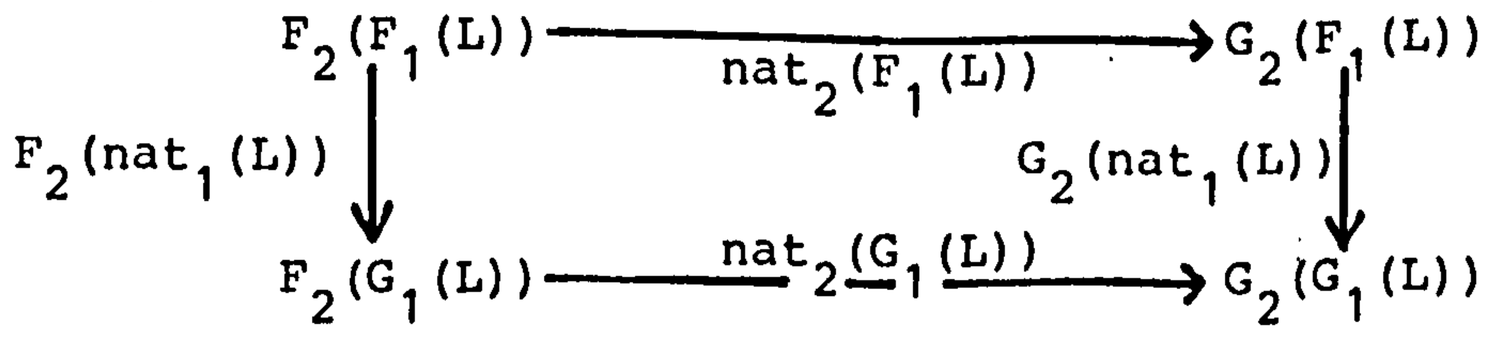
Fact 3.3:2. If nat_i are natural equivalences from F_i to G_i then $(nat_1,\ldots,nat_k)$ defined by

$$(nat_1,\ldots,nat_k)(L) = (nat_1(L),\ldots,nat_k(L))$$

is a natural equivalence from $(F_1,\ldots,F_k)$ to $(G_1,\ldots,G_k)$. ///

Fact 3.3:3. Let nat_i be natural equivalences from F_i to G_i and H and H' covariant functors. Then $H[nat_i]$ defined by $H[nat_i](L) = H(nat_i(L))$ is a natural equivalence from $H \bullet F_i$ to $H \bullet G_i$ and $nat_i[H']$ defined by $nat_i[H'](L) = nat_i(H'(L))$ is from $F_i \bullet H'$ to $G_i \bullet H'$. Furthermore $nat_2[G_1] \bullet F_2[nat_1]$ is a natural equivalence from $F_2 \bullet F_1$ to $G_2 \bullet G_1$ and it equals $G_2[nat_1] \bullet nat_2[F_1]$. ///

The equality in the last fact is illustrated by



and follows because nat_2 is a natural transformation. (In the terminology of category theory /ArMa75/ $nat_2 \cdot nat_1$ is the horizontal composition of nat_2 and nat_1 whereas $nat_2[G_1] \cdot F_2[nat_1]$ is the vertical composition.)

The connection between the bottom-level domains of $\underline{S}$ and $\underline{C}$ can now be stated.

Theorem 3.3:4. Assume that $V \vdash gt$, $V \vdash gt'$, $card(V) = N$ and that X has index 1 in V. Then there is a natural equivalence nat_{gt} from the functor

$$\rho \cdot \underline{S}[\![gt]\!] : \underline{\underline{ACCs}}^N \rightarrow \underline{\underline{ACLas}}$$

to

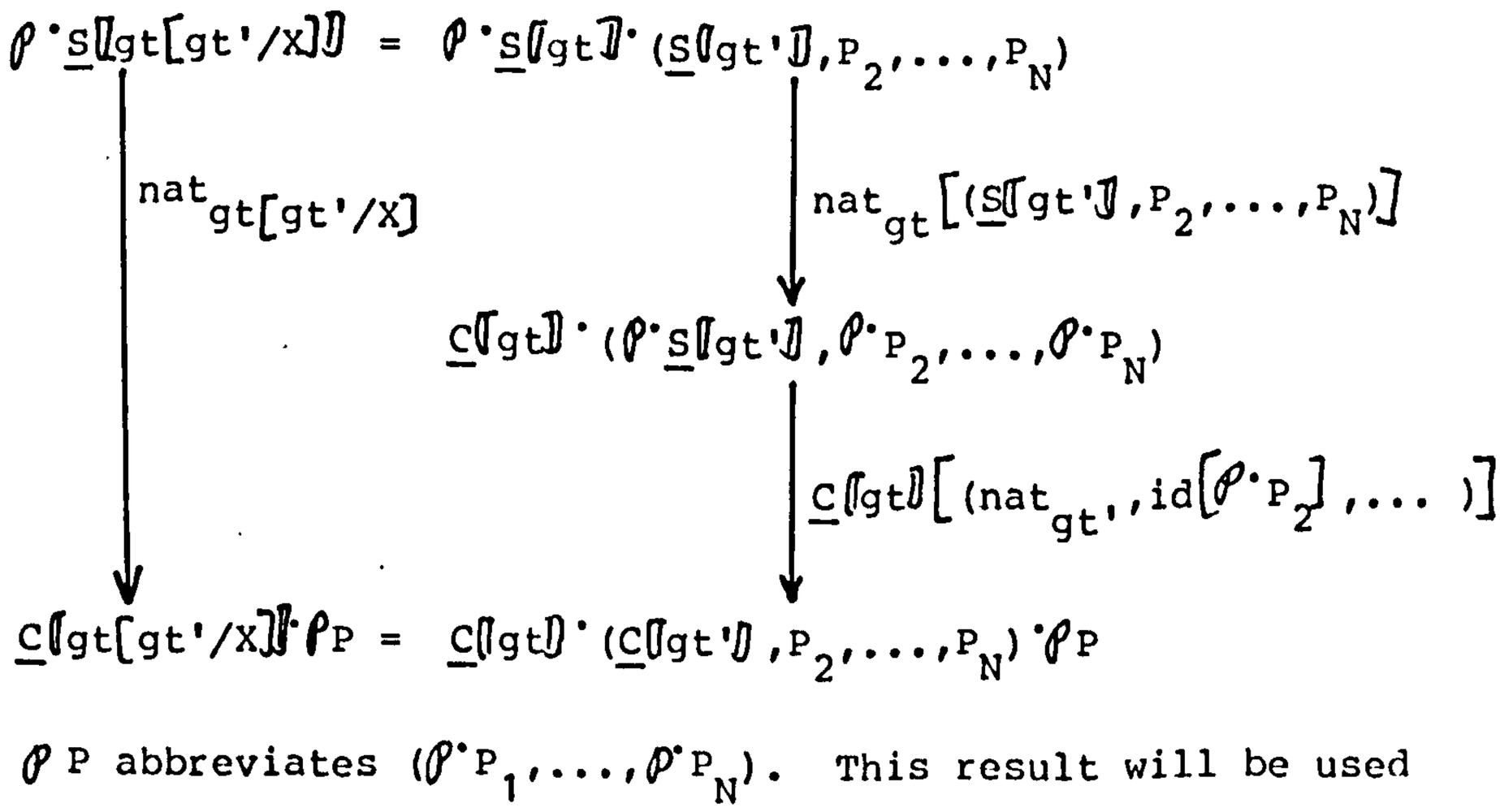
$$\underline{C}[\![gt]\!] \cdot (\rho \cdot P_1, \ldots, \rho \cdot P_N)$$

It satisfies that $nat_{gt[gt'/X]}$ equals

$$\underline{C}[\![gt]\!][(nat_{gt}, id[\rho \cdot P_2], \ldots, id[\rho \cdot P_N])] \cdot nat_{gt}[(\underline{S}[\![gt']\!], P_2, \ldots, P_N)]$$

///

Recalling lemma 2.3:2 the latter equality may be illustrated as the commutativity of



where ρP abbreviates $(\rho \cdot P_1, \ldots, \rho \cdot P_N)$. This result will be used later in lemma 3.3:11 and may be viewed as a companion to lemma 2.3:2.

Proof The proof of the theorem, and the construction of nat_{gt}, is by structural induction on gt.

Case $gt=\underline{A}_i$: The natural equivalence nat_{gt} will be chosen to be $nat_{gt}(D_1,\ldots,D_N) = id_{\rho(\underline{S}(\underline{A}_i))}$. This works because

$$\rho^{\cdot}\underline{S}[\![gt]\!] = \rho^{\cdot}\underline{S}(\underline{A}_i)^{\cdot}K_{NIL} = \rho^{\cdot}\underline{S}(\underline{A}_i)^{\cdot}K_{NIL}^{\cdot}\rho_P = \underline{C}[\![gt]\!]^{\cdot}\rho_P$$

For the desired equality between the natural equivalences one has

$$nat_{gt}(D_1,\ldots,D_N) = id_{\rho(\underline{S}(\underline{A}_i))}$$
$$C[\![gt]\!][\ldots](D_1,\ldots,D_N) = id_{\rho(\underline{S}(\underline{A}_i))}$$
$$nat_{gt}[\ldots](D_1,\ldots,D_N) = id_{\rho(\underline{S}(\underline{A}_i))}$$

and the result follows.

Case gt=Y. Assuming that the index of Y in V is i the natural equivalence is $nat_{gt} = id[\rho^{\cdot}P_i]$. Verification of the formula for $nat_{gt[gt'/X]}$ is straightforward: there are two cases depending on whether $X \equiv Y$ or not.

Case $gt=gt_1\underline{x}\ldots\underline{x}gt_k$. Let nat' be the natural equivalence from $\rho^{\cdot}x$ to $\otimes^{\cdot}(\rho^{\cdot}P_1,\ldots,\rho^{\cdot}P_k)$ guaranteed by theorem 3.2:16. Then define nat_{gt} as illustrated by

$$\begin{array}{ccc}
\rho^{\cdot}\underline{S}[\![gt]\!] & = & \rho^{\cdot}x^{\cdot}(\underline{S}[\![gt_1]\!],\ldots,\underline{S}[\![gt_k]\!]) \\
\Big\downarrow{\scriptstyle nat_{gt}} & & \Big\downarrow{\scriptstyle nat'[(\underline{S}[\![gt_1]\!],\ldots,\underline{S}[\![gt_k]\!])]} \\
 & & \otimes^{\cdot}(\rho^{\cdot}\underline{S}[\![gt_1]\!],\ldots,\rho^{\cdot}\underline{S}[\![gt_k]\!]) \\
 & & \Big\downarrow{\scriptstyle \otimes[(nat_{gt_1},\ldots,nat_{gt_k})]} \\
\underline{C}[\![gt]\!]^{\cdot}\rho_P & = & \otimes^{\cdot}(\underline{C}[\![gt_1]\!]^{\cdot}\rho_P,\ldots,\underline{C}[\![gt_k]\!]^{\cdot}\rho_P)
\end{array}$$

i.e. $nat_{gt} = \otimes[(nat_{gt_1},\ldots,nat_{gt_k})]^{\cdot}nat'[(\underline{S}[\![gt_1]\!],\ldots,\underline{S}[\![gt_k]\!])]$.

Verification of the formula for $nat_{gt[gt'/X]}$ is a straightforward but tedious calculation using that $\otimes$ is a functor over $\underline{\underline{\text{ACLas}}}$.

Case $gt=gt_1 \underline{*} \ldots \underline{*} gt_k$ is similar.

Case $gt=gt_1 \underline{+} \ldots \underline{+} gt_k$ is similar because a natural equivalence nat' from $\rho^{\cdot}+$ to $x^{\cdot}(\rho^{\cdot}P_1,\ldots,\rho^{\cdot}P_k)$ can be defined by

$$nat'(D_1,\ldots,D_k)(I) = (\{out_1(d) \mid d \in I\},\ldots,\{out_k(d) \mid d \in I\})$$

Case $gt=gt_0{}_{\underline{\perp}}$ is similar because a natural equivalence nat' from $\rho^{\cdot}\perp$ to $\perp^{\cdot}\rho$ can be defined by

$$nat'(D)(\{\perp_{(D_\perp)}\}) = \perp_{(\rho(D))_\perp}$$
$$nat'(D)(I) = up(\{down(d) \mid d \in I\}) \quad \text{when } I \neq \{\perp_{(D_\perp)}\}$$

Case $gt=\underline{rec}Y.gt_0$. To define nat_{gt} we shall assume that Y is an element of V and has index N. It is straightforward to adapt the definition if this is not so. Define the functors $F_{gt_0,n}$ and $G_{gt_0,n}$ inductively by

$$F_{gt_0,0}=K_U, \quad F_{gt_0,n+1}=\underline{S}[\![gt_0]\!]^{\cdot}(P_1,\ldots,P_{N-1},F_{gt_0,n})$$
$$G_{gt_0,0}=K_U, \quad G_{gt_0,n+1}=\underline{C}[\![gt_0]\!]^{\cdot}(P_1,\ldots,P_{N-1},G_{gt_0,n})$$

A natural equivalence $NAT_{gt_0,n}$ from $\rho^{\cdot}F_{gt_0,n}$ to $G_{gt_0,n}{}^{\cdot}\rho P$ is defined by

$$NAT_{gt_0,0}(D_1,\ldots,D_N) = \perp_{\rho(U)\to U}$$
$$NAT_{gt_0,n+1} = \underline{C}[\![gt_0]\!][(id[\rho^{\cdot}P_1],\ldots,NAT_{gt_0,n})]^{\cdot}nat_{gt_0}[(P_1,\ldots,F_{gt_0,n})]$$

Now consider $\underline{S}[\![gt]\!](D_1,\ldots,D_N)$ which is the object in the limiting cone $(D,(r_n)_n)$ for the chain

$$((F_{gt_0,n}(D_1,\ldots,D_N))_n\ ,\ (F_{gt_0,n}(id_{D_1},\ldots,id_{D_N}))_n)$$

as follows from the definition of REC_N in section 2.2. Similarly

$(D', (r'_n)_n)$ for $\underline{S}[\![gt]\!](D'_1,\ldots,D'_N)$

$(L, (s_n)_n)$ for $\underline{C}[\![gt]\!](\wp(D_1),\ldots,\wp(D_N))$

$(L', (s'_n)_n)$ for $\underline{C}[\![gt]\!](\wp(D'_1),\ldots,\wp(D'_N))$

The definition

$$nat_{gt}(D_1,\ldots,D_N) = \bigsqcup_n s_n \cdot NAT_{gt_0,n}(D_1,\ldots,D_N)\cdot\wp(r_n)^U$$

clearly gives an isomorphism because each $NAT_{gt_0,n}(D_1,\ldots,D_N)$ is an isomorphism.

To show that nat_{gt} is a natural equivalence note that $id_{\wp(D)} = \bigsqcup_n \wp(r_n)\cdot\wp(r_n)^U$ follows from sections 2.2 and 3.2: see e.g. 2.2:2 and recall that $\wp$ is a locally continuous functor. Considering $f_i:D_i\rightarrow D'_i$ it therefore suffices to show that

$$\underline{C}[\![gt]\!](\wp(f_1),\ldots)\cdot nat_{gt}(D_1,\ldots)\cdot\wp(r_n)$$

equals

$$nat_{gt}(D'_1,\ldots)\cdot\wp(\underline{S}[\![gt]\!](f_1,\ldots))\cdot\wp(r_n)$$

for all values of n. Using the definitions of $\underline{C}[\![gt]\!]$, $\underline{S}[\![gt]\!]$ and nat_{gt} lemmas 2.2:11 and 2.2:6 can be used to reduce this to the desired equality of

$$s'_n\cdot G_{gt_0,n}(\wp(f_1),\ldots)\cdot NAT_{gt_0,n}(D_1,\ldots)$$

and

$$s'_n\cdot NAT_{gt_0,n}(D'_1,\ldots)\cdot\wp(F_{gt_0,n}(f_1,\ldots))$$

This result is immediate, however, because $NAT_{gt_0,n}$ is a natural equivalence from $\wp\cdot F_{gt_0,n}$ to $G_{gt_0,n}\cdot\wp P$.

It remains to verify the formula for $nat_{gt[gt'/X]}$. If X and Y are the same variable then this is straightforward so assume otherwise. We shall assume that Y is not free in gt' (otherwise renaming could

achieve this). First we shall see that a similar formula holds for each $NAT_{gt_0[gt'/X],n}$ namely that

$$NAT_{gt_0[gt'/X],n} =$$
$$G_{gt_0,n}[(nat_{gt'},id[\rho\cdot P_2],\ldots,id[\rho\cdot P_N])]$$
$$\cdot\ NAT_{gt_0,n}[(\underline{S}[gt'](),P_2,\ldots,P_N)]$$

This can be verified by induction on n. In each case both sides are instantiated at $(D_1,\ldots,D_N)$ and the case n=0 is immediate. The proof in the inductive-case is by a lengthy calculation which will be omitted except for mentioning that the following facts are used in the calculation:

- lemma 2.3:2 and $F_{gt_0[gt'/X],n} = F_{gt_0,n}\cdot(\underline{S}[gt'](),P_2,\ldots,P_N)$
- the formula for $nat_{gt_0[gt'/X]}$
- Y is not free in gt' so $\underline{C}[gt']()$, $\underline{S}[gt']()$ and $nat_{gt'}$ do not depend on the N'th argument
- $\underline{C}[gt_0]()$ and $\underline{C}[gt']()$ are functors
- the inductive hypothesis in the induction on n.

Turning to the formula for $nat_{gt[gt'/X]}$ we have the equality

$$nat_{gt[gt'/X]}(D_1,\ldots,D_N)$$
$$= \bigcup_n s_n\cdot NAT_{gt_0[gt'/X],n}(D_1,\ldots,D_N)\cdot\rho(r_n)^U$$

Write $L_i = \rho(D_i)$ and assume N = 2 below. By lemma 2.3:2 and $G_{gt_0[gt'/X],n} = G_{gt_0,n}\cdot(\underline{C}[gt'](),P_2,\ldots,P_N)$ the functionalities are

$$s_n : G_{gt_0,n}(\underline{C}[gt'](L_1,L_N),L_N) \to \underline{C}[gt](\underline{C}[gt'](L_1,L_N),L_N)$$
$$r_n: F_{gt_0,n}(\underline{S}[gt'](D_1,D_N),D_N) \to \underline{S}[gt](\underline{S}[gt'](D_1,D_N),D_N)$$

If similarly

$$s''_n : G_{gt_0,n}(\rho(\underline{S}[gt'](D_1,D_N)),L_N) \to \underline{C}[gt](\rho(\underline{S}[gt'](D_1,D_N)),L_N)$$

we have

$$\underline{C}[\![gt]\!]\,[(nat_{gt'}, id[\rho\cdot P_2], \ldots)]\,(D_1, \ldots, D_N)$$
$$= \sqcup_n\ s_n \cdot G_{gt_0,n}(nat_{gt'}(D_1, \ldots, D_N), id_{\rho(D_2)}, \ldots) \cdot s''^{U}_n$$

and

$$nat_{gt}[(\underline{S}[\![gt']\!], P_2, \ldots)]\,(D_1, \ldots, D_N)$$
$$= \sqcup_n\ s''_n \cdot NAT_{gt_0,n}(\underline{S}[\![gt']\!](D_1, \ldots, D_N), D_2, \ldots) \cdot \rho(r_n)^U$$

By the formula for $NAT_{gt_0[gt'/X],n}$ and lemma 2.2:6 the formula for $nat_{gt[gt'/X]}$ follows. ///

This theorem does not generalise to top-level types. This is immediate for a type t such that P(∅,t) (see section 2.1) as $\underline{S}[\![t]\!]$ then equals $\underline{C}[\![t]\!]$. Also in the case of $gt \underline{\to} gt'$ one has $\rho(\underline{S}[\![gt \underline{\to} gt']\!])$ to be of the form $\rho(X \to Y)$ whereas $\underline{C}[\![gt \underline{\to} gt']\!]$ is of the form $\rho(X) \to \rho(Y)$. In a sense $\underline{\to}$ is treated in an independent attribute method rather than a relational method as will be discussed in chapter 6.

EXPRESSION PART

We now turn to defining the expression part of $\underline{C}$. The transformations strict, lin and $view_t$ that are used in the definition will be explained afterwards. The first of the three components specifies the functions

$$\underline{C}(take_i) = \lambda 1.id^{x}(1)\downarrow i$$
$$\underline{C}(smashtake_i) = \lambda 1.id^{*}(1)\downarrow i$$
$$\underline{C}(in_i) = \lambda 1.(\perp, \ldots, 1, \ldots, \perp)$$
$$\underline{C}(up) = \lambda 1.(0,1)$$
$$\underline{C}(fold) = \theta \quad \text{-- see below}$$
$$\underline{C}(unfold) = \theta^{-1} \quad \text{-- see below}$$

where θ is the isomorphism from $\underline{C}[\![gt[\underline{rec}X.gt/X]]\!]$ to $\underline{C}[\![\underline{rec}X.gt]\!]$. (As was discussed in section 2.3 the intended gt is left implicit.)

The second component specifies the functionals

$$\underline{C}(\text{tuple})(g_1,\ldots,g_k) = \text{lin}(\text{cross}_x{}^{\cdot}\lambda 1.(g_1(1),\ldots,g_k(1)))$$

$$\underline{C}(\text{smashtuple})(g_1,\ldots,g_k) =$$

$$\text{lin}(\text{cross}_*{}^{\cdot}\text{smash}{}^{\cdot}\lambda 1.(g_1(1),\ldots,g_k(1)))$$

$$\underline{C}(\text{case})(g_1,\ldots,g_k) = \lambda 1.g_1(1\downarrow 1)\cup\ldots\cup g_k(1\downarrow k)$$

$$\underline{C}(\text{lift})(g) = \lambda 1.\text{def}(1)\rightarrow g(\text{down}(1)),\perp$$

$$\underline{C}(\text{cond})(g_1,g_2,g_3) =$$

$$\text{strict}(g_2){}^{\cdot}\text{filter}_{tt}(g_1)\cup\text{strict}(g_3){}^{\cdot}\text{filter}_{ff}(g_1)$$

$$\underline{\text{where}}\ \text{filter}_x(g) = \text{lin}(\lambda 1.g(1)\sqsupseteq\{x\}_R\rightarrow 1,\perp)$$

$$\underline{C}(\square)(g_1,g_2) = g_1{}^{\cdot}g_2$$

As an example $\underline{C}$(cond) will be explained below. Finally, the third component defines

$$\underline{C}(f) = \text{view}_t()(\underline{S}(f))$$

for each constant f of contravariantly pure type t.

The transformation $\underline{\text{lin}}$ is defined for any $\underline{\underline{\underline{ACL}}}$ morphism $g:L\rightarrow M$ by the formula

$$\text{lin}(g) = \lambda 1.\bigsqcup\{g(i) \mid i\sqsubseteq 1 \wedge i\in IB_L\}$$

which clearly gives a continuous function and depends continuously on g. For the collecting interpretation one is especially interested in the case where L (and M) are powerdomains. Then

$$\text{lin}(g) = \lambda 1.\bigcup\{g(i) \mid i\sqsubseteq 1 \wedge i\in PB_L\} = (g{}^{\cdot}\lambda d.\{d\}_R)^{\dagger}$$

follows from theorem 3.1:7. It is seen that lin(g) is an additive function that agrees with g upon singletons and it is this that motivates the use of lin (as will become clear when filter_x is explained below). Of these three characterisations of lin only the first two work for all $\underline{\underline{\underline{ACL}}}$ morphisms and the definition chosen is

the one that will be useful in section 4.4.

Next consider $\text{filter}_x(g)$. It follows from the type considerations of section 2.1 that g will be an element of $\underline{C}[\![gt \underline{\rightarrow} T]\!]$ for some gt, i.e. $g: L \rightarrow \wp(T_\perp)$ for some $\underline{\underline{ACL}}$ object L. By theorem 3.3:4 there will be a powerdomain $\wp(D)$ that is isomorphic to L so let us assume that $L = \wp(D)$. Then for x=tt or x=ff we have

$$\text{filter}_x(g)(I) = LC(\{\perp\} \cup \{b \in I \mid g(\{b\}_R) \ni x\})$$

and in the case where $g = \wp(f)$ this gives

$$\text{filter}_x(\wp(f))(I) = LC(\{\perp\} \cup \{b \in I \mid f(b) = x\})$$

so that $\text{filter}_{tt}(g)(I)$ specifies the subset of I for which the first branch is appropriate and similarly for $\text{filter}_{ff}(g)(I)$. It is straightforward that $\text{filter}_x(g)$ is continuous and is continuous in g.

The conditional then executes each branch with the appropriate set of arguments. So the idea is that

$$\wp(\underline{S}(\text{cond})(f_1, f_2, f_3)) = \underline{C}(\text{cond})(\wp(f_1), \wp(f_2), \wp(f_3))$$

which agrees with what was said in the introduction. For this to hold it is necessary to use the continuous function strict defined by

$$\text{strict}(g) = \lambda l.\ l = \perp \rightarrow \perp, g(l)$$

unless f_2 and f_3 are both strict. (This relates to the remark in section 2.3.) Clearly strict(g) is a strict and continuous function that agrees with g except possibly on $\perp$.

Finally the transformation $\underline{\text{view}}_t()$ used for the third component must be specified. Consider as an example a function $f \in \underline{S}[\![gt \underline{\rightarrow} gt']\!]$

so that $f:\underline{S}[\![gt]\!] \to \underline{S}[\![gt']\!]$. Had it been the case that $\underline{C}[\![gt]\!] = \rho(\underline{S}[\![gt]\!])$ and likewise for gt' then the desired function would be

$$\rho(f) = \lambda I.LC(\{f(b) \mid b \in I\})$$

This is not quite the case but by theorem 3.3:4, and recalling that $\emptyset \vdash gt$ and $\emptyset \vdash gt'$, we can use

$$vw_{gt \to gt'}(f) = nat_{gt'}() \cdot \rho(f) \cdot nat_{gt}()^{-1}$$

instead. It is now necessary to extend this definition to all constants of contravariantly pure type. This will be accomplished by the continuous function

$$view_t() : \underline{S}[\![t]\!] \to \underline{C}[\![t]\!]$$

It will be defined structurally on t so types will be met that are not contravariantly pure but only CP(V,t) for some V such that $V \vdash t$. When $card(V) = N$ and $h_i:D_i \to L_i$ are morphisms of $\underline{\underline{CPO}}$ we therefore have

$$view_t(h_1,\ldots,h_N) : \underline{S}[\![t]\!](D_1,\ldots,D_N) \to \underline{C}[\![t]\!](L_1,\ldots,L_N)$$

Clearly $view_{ft}()$ is to be vw_{ft} but it will sometimes be convenient to make this explicit by writing

$$view_t[(vw_{ft})_{ft}](h_1,\ldots,h_N)$$

This makes it possible to perform the development in sufficient generality that it is not necessary to give a similar development in chapter 4.

The definition of $view_t$ closely resembles the definition of the functor $[\![t]\!]$ in section 2.3 (see theorem 2.2:10 for recX.t). It is worth repeating that it is assumed that $V \vdash t$, CP(V,t) and card(V)=N.

$$\mathrm{view}_{A_i}[vw](h_1,\ldots,h_N)(v) = v$$

$$\mathrm{view}_{t_1\times\ldots\times t_k}[vw](h_1,\ldots,h_N)(v) =$$
$$(\mathrm{view}_{t_1}[vw](h_1,\ldots,h_N)(v\downarrow 1),\ldots,\mathrm{view}_{t_k}[vw](h_1,\ldots,h_N)(v\downarrow k))$$

$$\mathrm{view}_{t_1 * \ldots * t_k}[vw](h_1,\ldots,h_N)(v) =$$
$$\mathrm{smash}(\mathrm{view}_{t_1}[vw](h_1,\ldots,h_N)(v\downarrow 1),\ \ldots\ )$$

$$\mathrm{view}_{t_1+\ldots+t_k}[vw](h_1,\ldots,h_N)(v) =$$
$$\mathrm{is}_1(v) \rightarrow \mathrm{in}_1(\mathrm{view}_{t_1}[vw](h_1,\ldots,h_N)(\mathrm{out}_1(v))),$$
$$\ldots \quad ,\ldots$$
$$\ldots \quad ,\bot$$

$$\mathrm{view}_{t_\perp}[vw](h_1,\ldots,h_N)(v) =$$
$$\mathrm{def}(v) \rightarrow \mathrm{up}(\mathrm{view}_t[vw](h_1,\ldots,h_N)(\mathrm{down}(v))),\bot$$

$$\mathrm{view}_{t_1\rightarrow t_2}[vw](h_1,\ldots,h_N)(v) =$$
$$\lambda x.\mathrm{view}_{t_2}[vw](h_1,\ldots,h_N)(v(x))$$

$$\mathrm{view}_{\mathrm{recX}.t}[vw](h_1,\ldots,h_N)(v) =$$

$$\begin{cases} v & \text{if } P(V,\mathrm{recX}.t) \\ \mathrm{LFP}(\lambda h.\Theta'\cdot\mathrm{view}_t[vw](h_1,\ldots,h_N,h)\cdot\Theta^{-1})(v) & \text{otherwise} \end{cases}$$

where Θ' and Θ are the obvious isomorphisms and assuming that X has index N+1 in $V\upsilon\{X\}$

$$\mathrm{view}_X[vw](h_1,\ldots,h_N)(v) = h_i(v)$$

assuming that X has index i in V

$$\mathrm{view}_{gt\underline{\rightarrow}gt'}[vw](h_1,\ldots,h_N)(v) = vw_{gt\underline{\rightarrow}gt'}(v)$$
(presently $\mathrm{nat}_{gt'}()\cdot(v)\cdot\mathrm{nat}_{gt}()^{-1}$)

That this definition makes sense is by a structural induction showing that $\mathrm{view}_t[vw](h_1,\ldots,h_N)$ is continuous and depends continuously on h_i (provided that vw_{ft} and h_i are all continuous). In the case $t_1\rightarrow t_2$ lemma 2.3:3 is needed to show that $P(V,t_1)\wedge V\vdash t_1$ guarantees that $\underline{S}[\![t_1]\!](D_1,\ldots,D_N)$ equals $\underline{C}[\![t_1]\!](L_1,\ldots,L_N)$. It is to achieve this that the constants are required to be of contravariantly pure type for when

$\underline{S}[\![t_1]\!](D_1,\ldots,D_N)$ differs from $\underline{C}[\![t_1]\!](L_1,\ldots,L_N)$ it is not clear how to define $view_{t_1\to t_2}$. In the case recX.t lemma 2.3:2 shows that the functionality is correct.

The resemblance of $view_t[vw]$ to the functor $[\![t']\!]$ defined in section 2.3 can be formalised in the following way.

Lemma 3.3:5. Let t be a type such that $V \vdash t$, $CP(V,t)$ and $card(V)=N$. Let $ft_1, \ldots, ft_n$ be all the ft-types mentioned in t and let $t'=t[X_{N+i}/ft_i]$ have each ft_i replaced by X_{N+i}. Then for all choices of CPOs morphisms h_i, vw_{ft_j} and g_{N+j} and for all interpretations $\underline{I}$ it is the case that

$$view_t[vw](h_1,\ldots,h_N)$$

equals

$$\underline{I}[\![t']\!]((h_1,g_1),\ldots,(h_N,g_N),(vw_{ft_1},g_{N+1}),\ldots,(vw_{ft_n},g_{N+n}))\downarrow 1 \quad ///$$

Proof This is a structural induction and we sketch two cases. Case $t=t_1\to t_2$ uses lemma 2.3:3 and that $\underline{I}[\![t_1']\!]$ is a functor to show that

$$\underline{I}[\![t_1']\!]((h_1,g_1),\ldots,(vw_{ft_n},g_{N+n}))\downarrow 1 = id$$

Case $t=recX.t_0$ uses lemma 2.2:10 in the case where $P(V,recX.t_0)$ fails and is straightforward in the case where it holds. ///

This result makes it immediate to prove the following property of $view_t[vw]$. Call a function h strongly strict iff $h(v)=\perp \Leftrightarrow v=\perp$.

Lemma 3.3:6. If vw_{ft} and h_i are all strongly strict then so is $view_t[vw](h_1,\ldots,h_N)$. ///

Proof This is a structural induction and the only non-trivial case is $t=recX.t_0$ when $P(V,recX.t_0)$ fails. From the previous lemma it is not

difficult to see that

$$\mathrm{view}_{\mathrm{rec}X.t_0}[\![vw]\!](h_1,\ldots,h_N) =$$
$$\bigsqcup_n s_n \cdot (\lambda h.\mathrm{view}_{t_0}[\![vw]\!](h_1,\ldots,h_N,h))^n(\bot) \cdot r_n^U$$

for appropriate embeddings s_n and r_n in $\underline{\underline{\mathrm{CPOs}}}$. Since $\bot:U\to U$ is strongly strict and it easily follows that also all

$$(\lambda h.\mathrm{view}_{t_0}[\![vw]\!](h_1,\ldots,h_N,h))^n(\bot)$$

are we get that $\mathrm{view}_{\mathrm{rec}X.t_0}[\![vw]\!](h_1,\ldots,h_N)$ is strongly strict. ///

A consequence of this lemma is that the smash in $\mathrm{view}_{t_1*\ldots*t_k}$ is not always necessary. It will emerge shortly that strongly strictness is only of interest because of the presence of the domain constructor $*$ (see a remark in section 2.1).

Returning to the definition of the expression part of $\underline{C}$ it can be seen that is satisfies the requirements of section 2.3: For $\underline{C}(\mathrm{in}_1)$ etc. in the first component this is a straightforward verification using lemma 2.3:2 for $\underline{C}$(fold) and $\underline{C}$(unfold). The functionals are clearly continuous and continuous in the g_i arguments. Also the constants of contravariantly pure type are as required. In the remainder of this section we therefore consider the relationship between the semantics of expressions as given by $\underline{S}$ and $\underline{C}$.

Relating the interpretations S and C

In the explanation of $\underline{C}$(cond) it was hinted at the desired connection between the denotations for expressions in the standard and collecting semantics. The formal statement of this connection amounts to defining a predicate $\mathrm{sim}_t()$ on $\underline{S}[\![t]\!]\times\underline{C}[\![t]\!]$. As for view_t it is convenient to consider

$$sim_t[(sm_{ft})_{ft}](Q_1,\ldots,Q_N)$$

that is a predicate on $\underline{S}[\![t]\!](D_1,\ldots,D_N)\times\underline{C}[\![t]\!](L_1,\ldots,L_N)$ whenever each Q_i is a predicate on $D_i\times L_i$ and sm_{ft} on $\underline{S}[\![ft]\!]\times\underline{C}[\![ft]\!]$. It is assumed that $V \vdash t$ and $card(V)=N$.

The predicate is defined structurally much as was the case for $view_t$.

$$sim_{A_i}[sm](Q_1,\ldots,Q_N)(v,w) \equiv v=w$$

$$sim_{t_1\times\ldots\times t_k}[sm](Q_1,\ldots,Q_N)(v,w) \equiv$$
$$\forall i:\ sim_{t_i}[sm](Q_1,\ldots,Q_N)(v\downarrow i, w\downarrow i)$$

$$sim_{t_1 * \ldots * t_k}[sm](Q_1,\ldots,Q_N)(v,w) \equiv$$
$$\forall i:\ sim_{t_i}[sm](Q_1,\ldots,Q_N)(v\downarrow i, w\downarrow i)$$

$$sim_{t_1+\ldots+t_k}[sm](Q_1,\ldots,Q_N)(v,w) \equiv (v=\perp\wedge w=\perp)\ \vee$$
$$\exists i:\ v=(i,v')\wedge\ w=(i,w')\wedge sim_{t_i}[sm](Q_1,\ldots,Q_N)(v',w')$$

$$sim_{t_\perp}[sm](Q_1,\ldots,Q_N)(v,w) \equiv (v=\perp\wedge w=\perp)\ \vee$$
$$v=up(v')\wedge\ w=up(w')\wedge\ sim_t[sm](Q_1,\ldots,Q_N)(v',w')$$

$$sim_{t_1\to t_2}[sm](Q_1,\ldots,Q_N)(v,w) \equiv$$
$$\forall(v',w'):\ sim_{t_1}[sm](Q_1,\ldots,Q_N)(v',w')$$
$$\Rightarrow sim_{t_2}[sm](Q_1,\ldots,Q_N)(v(v'),w(w'))$$

$$sim_{recX.t}[sm](Q_1,\ldots,Q_N)(v,w) \equiv$$
$$\forall n:\ SIM_{t,n}[sm](Q_1,\ldots,Q_N)(r_n^U(v),s_n^U(w))$$

where $SIM_{t,0}[sm](Q_1,\ldots,Q_N)(v',w')$ is true

$$SIM_{t,n+1}[sm](Q_1,\ldots,Q_N) =$$
$$sim_t[sm](Q_1,\ldots,Q_N, SIM_{t,n}[sm](Q_1,\ldots,Q_N))$$

and it is assumed that X has index N+1 in $V\cup\{X\}$ and that r_n and s_n are the CPOs embeddings of the respective limiting cones

$$sim_X[sm](Q_1,\ldots,Q_N)(v,w) \equiv Q_i(v,w) \qquad \text{if X has index i in V}$$

$$sim_{gt\to gt'}[sm](Q_1,\ldots,Q_N)(v,w) \equiv sm_{gt\to gt'}(v,w)$$

The sm that is of interest in this section is

$$sm_{gt \underline{\rightarrow} gt'}(v,w) \equiv w = nat_{gt'}() \cdot \rho(v) \cdot nat_{gt}()^{-1}$$

It is a straightforward structural induction to prove that this definition gives rise to a predicate as stated. Furthermore it is convenient to state the following information about the predicate. First an analogue of lemma 3.3:6:

...

Lemma 3.3:7. If each sm'_{ft} and Q_i is an admissible predicate then also $sim_t[sm'](Q_1,\ldots,Q_N)$ is. If further sm'_{ft} and Q_i satisfies that

when true of (v,w) then $v=\perp$ iff $w=\perp$ (⋇)

then also $sim_t[sm'](Q_1,\ldots,Q_N)$ does. ///

The proof is by structural induction with an additional numerical induction for the case recX.t. Both admissibility and (⋇) hold for sm_{ft} and as with "strongly strict" the property (⋇) is only of interest because of the domain constructor ⋇. Next an analogue of lemma 2.3:2 is:

Lemma 3.3:8. If $V \vdash t$, $V \vdash t'$, card(V)=N and X has index N in V then

$$sim_{t[t'/X]}[sm'](Q_1,\ldots,Q_N) =$$
$$sim_t[sm'](Q_1,\ldots,Q_{N-1},sim_t,[sm'](Q_1,\ldots,Q_N)) \quad ///$$

This result is proved by induction as for the previous lemma. Finally an analogue of lemma 2.3:3 is:

Lemma 3.3:9. If $V \vdash t$, P(V,t) and card(V)=N then

$sim_t[sm'](Q_1,\ldots,Q_N)(v,w)$ holds iff v=w ///

Proof This is a corollary of the following result that is proved by structural induction on t.

Let V⊢t, P(V',t), card(V)=N, card(V')=n be such that V' is the subset of variables in V with index at most n in V. If eqv(v,w) is defined by v=w then $sim_t[sm'](Q_1,\ldots,Q_n,eqv,\ldots,eqv) = eqv$.

We omit the proof. ///

...

The purpose of $view_t$ is to transform a function into one that is related to it by sim_t. That this holds is made precise in:

Lemma 3.3:10. Considering a contravariantly pure and closed type t we have

$$sim_t()(f,view_t()(f))$$

for every $f \in S[\![t]\!]$. In general

$$sim_t[sm']()(f,view_t[vw']()(f))$$

holds provided that $sm'_{ft}(f',vw'_{ft}(f'))$ is guaranteed and all vw'_{ft} are strongly strict. ///

Proof Clearly the explicitly defined sm and vw satisfy the conditions stated for sm' and vw'. The lemma then follows from the more general result that

V ⊢ t and CP(V,t) and card(V) = N implies

$sim_t[sm'](Q_1,\ldots,Q_N)(f,view_t[vw'](h_1,\ldots,h_N)(f))$

whenever $sm'_{ft}(f',vw'_{ft}(f'))$ and $Q_i(f',h_i(f'))$ are guaranteed and vw'_{ft} and h_i are strongly strict.

This result is proved by structural induction on t such that V ⊢ t and CP(V,t).

Case $t=A_i$ is immediate.

Case $t=t_1\times\ldots\times t_k$ is straightforward from the inductive hypotheses.

Case $t=t_1*\ldots*t_k$ is as above because lemma 3.3:6 shows that smash has no effect.

Case $t=t_1+\ldots+t_k$ and case $t=t_{0\perp}$ are straightforward.

Case $t=t_1\to t_2$ is straightforward because lemma 3.3:9 shows that $sim_{t_1}[\ldots](\ldots)(v',w')$ holds iff $v'=w'$.

Case $t=recX.t_0$. There are two cases. If $P(V,recX.t_0)$ holds then lemma 3.3:9 shows that $sim_t[\ldots](\ldots)(v',w')$ holds if $v'=w'$. Since $view_t[vw'](h_1,\ldots,h_N)(v)=v$ the result is immediate. Suppose next that $P(V,recX.t_0)$ is false. The formulation of $view_t$ given in the proof of lemma 3.3:6 makes it convenient to abbreviate (assuming that X has index N+1 in $V\cup\{X\}$)

$$H(h) = view_{t_0}[vw'](h_1,\ldots,h_N,h)$$

It then follows that (e.g. by lemmas 3.3:5 and 2.2:11)

$$s_n^U(view_t[\ldots](\ldots)(v)) = H^n(\perp)(r_n^U(v))$$

Since $\perp:U\to U$ is strongly strict also $H^n(\perp)$ is and it suffices to prove that

$$SIM_{t_0,n}[sm'](Q_1,\ldots,Q_N)(\ r_n^U(v)\ ,\ H^n(\perp)(r_n^U(v))\ )$$

This is by induction on n using the hypotheses of the structural induction.

Cases t=X and t=ft are immediate. ///

We are now ready to prove a relationship between the expression parts of the standard and collecting interpretations. Henceforth [vw] and [sm] will be left implicit for $view_t()$ and $sim_t()$.

Lemma 3.3:11. For every entry z of type t in one of the components of the expression part of an interpretation we have $sim_t()(\underline{S}(z),\underline{C}(z))$.///

Proof There are three components in the expression part and by lemma 3.3:10 the result is immediate for the third component. For the other two components this is by explicit calculations. In these the parentheses are omitted from the natural equivalences nat_{gt} because $\emptyset \vdash gt$.

...

The calculation for $take_i$ starts with

$$nat_{gt_i}^{-1} \cdot \underline{C}(take_i) \cdot nat_{gt_1} \underline{\times} \ldots \underline{\times} gt_k$$

The definition of nat in theorem 3.3:4 then gives

$$nat_{gt_i}^{-1} \cdot \downarrow i \cdot id^{\times} \cdot nat_{gt_1} \otimes \ldots \otimes nat_{gt_k} \cdot nat'(\underline{S}[\![gt_1]\!], \ldots, \underline{S}[\![gt_k]\!])$$

and the definition of the functor $\otimes$ gives

$$\downarrow i \cdot (nat_{gt_1}^{-1} \times \ldots) \cdot id^{\times} \cdot (cross \cdot (nat_{gt_1} \times \ldots))^{\times} \cdot nat'(\underline{S}[\![gt_1]\!], \ldots)$$

Since $f^{\times} \cdot cross = f$ and $g \cdot f^{\times} = (g \cdot f)^{\times}$ whenever g is additive this gives

$$\downarrow i \cdot ((nat_{gt_1}^{-1} \times \ldots) \cdot (nat_{gt_1} \times \ldots))^{\times} \cdot nat'(\underline{S}[\![gt_1]\!], \ldots)$$

and because $\times$ is a functor this is

$$\downarrow i \cdot id^{\times} \cdot nat'(\underline{S}[\![gt_1]\!], \ldots)$$

But nat' was constructed as an isomorphism between two tensor products (theorem 3.2:16) so this equals

$$\downarrow i \cdot \lambda I.(\{b\downarrow 1 \mid b \in I\}, \ldots, \{b\downarrow k \mid b \in I\})$$

using fact 3.2:4 and the extension of functions for the other tensor product. It is immediate that this equals

$$\wp(\underline{S}(take_i))$$

The calculations for $smashtake_i$, in_i and up follow this pattern.

Next consider the calculation for unfold. The desired result

$$\underline{C}(\text{unfold}) = \text{nat}_{gt[\underline{rec}X.gt/X]} \cdot \mathcal{P}(\underline{S}(\text{unfold})) \cdot \text{nat}_{\underline{rec}X.gt}^{-1}$$

is by theorem 3.3:4 equivalent to

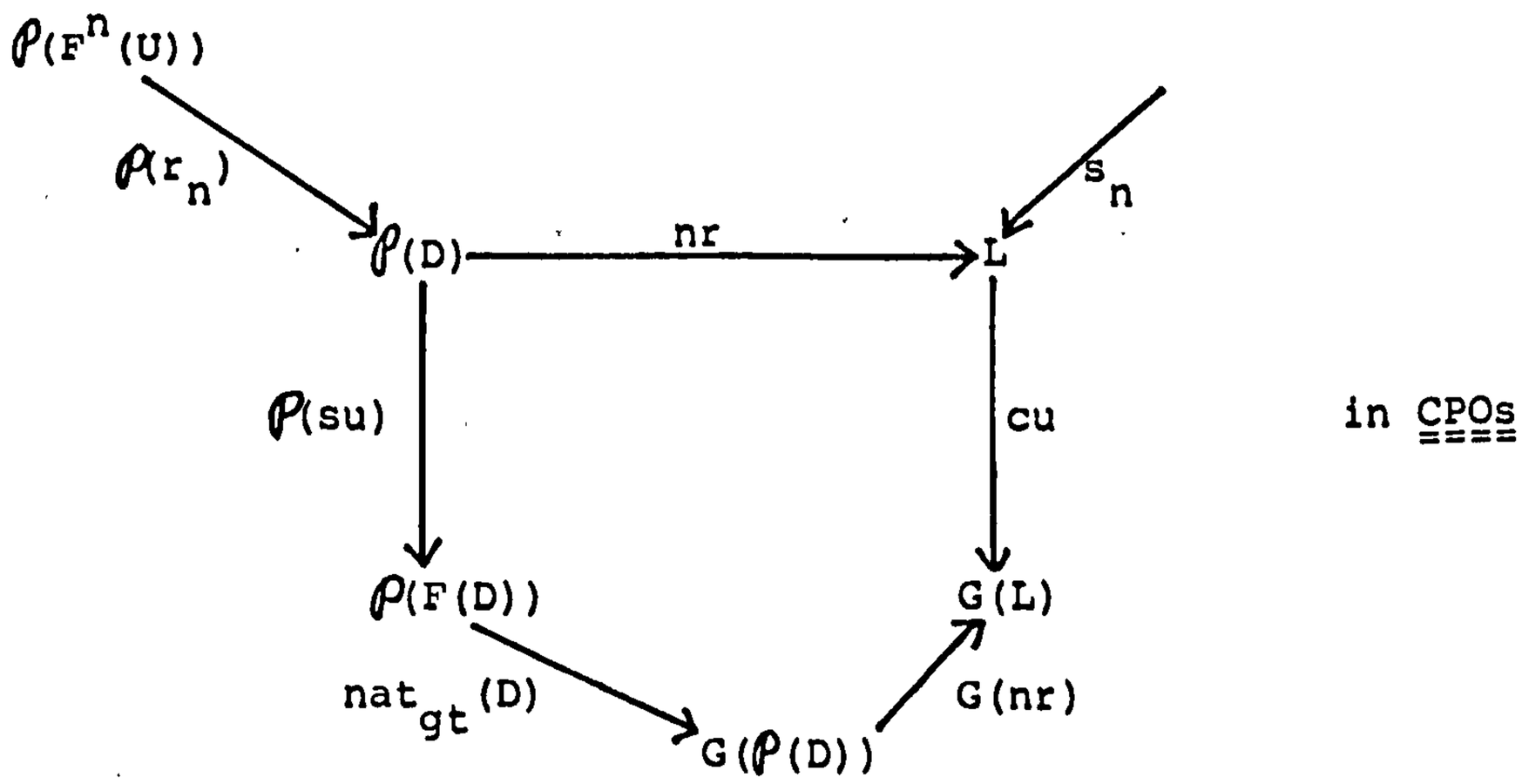
$$\underline{C}(\text{unfold}) \cdot \text{nat}_{\underline{rec}X.gt}$$
$$= \underline{C}[\![gt]\!](\text{nat}_{\underline{rec}X.gt}) \cdot \text{nat}_{gt}(\underline{S}[\![\underline{rec}X.gt]\!]) \cdot \mathcal{P}(\underline{S}(\text{unfold}))$$

which will be abbreviated to

$$\ldots$$

$$cu \cdot nr = G(nr) \cdot \text{nat}_{gt}(D) \cdot \mathcal{P}(su)$$

Because $\text{id}_{\mathcal{P}(D)} = \bigsqcup_n \mathcal{P}(r_n) \cdot \mathcal{P}(r_n)^U$ for embeddings r_n of $\underline{\underline{\text{CPO}}}$ given as in section 2.2 this follows if



in $\underline{\underline{\text{CPOs}}}$

can be shown to commute where $F=\underline{S}[\![gt]\!]$ and $L=\underline{C}[\![\underline{rec}X.gt]\!]$. This is immediate for n=0 and for n>0 it follows from the following calculation. It makes use of the definition of nat in theorem 3.3:4 and also theorem 2.2:4. For the upper path

$$cu \cdot nr \cdot \mathcal{P}(r_n) = G(s_{n-1}) \cdot NAT_{gt,n}$$

where $NAT_{gt,n}$ is defined in theorem 3.3:4. For the lower path

$$G(nr) \cdot \text{nat}_{gt}(D) \cdot \mathcal{P}(su) \cdot \mathcal{P}(r_n) =$$
$$G(nr) \cdot \text{nat}_{gt}(D) \cdot \mathcal{P}(F(r_{n-1})) =$$

by nat_{gt} a natural equivalence

$$G(nr)\cdot G(\mathcal{P}(r_{n-1}))\cdot nat_{gt}(F^{n-1}(U)) =$$

and by G a functor

$$G(s_{n-1})\cdot G(NAT_{qt,n-1})\cdot nat_{gt}(F^{n-1}(U))$$

But by the definition of NAT this is the result along the upper path.

Finally, the result for fold is immediate from the result for unfold.

It remains to verify the connection between the functionals. The calculation for cond has been sketched previously (under the assumption that the natural equivalences are the identities). The calculation for tuple is illustrative for the other functionals as well and is the only one that will be sketched. Consider functions $f_i \in \underline{S}[\![gt \to gt_i]\!]$ and abbreviate

$$ct = \underline{C}(tuple)(view_{gt\to gt_1}()(f_1),\ldots,view_{gt\to gt_k}()(f_k))$$
$$= lin(cross\cdot\lambda l.((nat_{gt_1}\cdot\mathcal{P}(f_1)\cdot nat_{gt}^{-1})(l),\ldots)$$

and

$$st = view_{gt\to gt_1\underline{x}\ldots\underline{x}gt_k}()(\underline{S}(tuple)(f_1,\ldots,f_k))$$
$$= nat_{gt_1\underline{x}\ldots\underline{x}gt_k}\cdot\mathcal{P}(\lambda d.(f_1(d),\ldots,f_k(d)))\cdot nat_{gt}^{-1}$$

Both ct and st are continuous and additive so it suffices to consider a prime basis element p of $\underline{C}[\![gt]\!]$ and show that ct(p) = st(p). Defining $x_i = \mathcal{P}(f_i)(nat_{gt}^{-1}(p))$ this gives

$$ct(p) = cross(nat_{gt_1}(x_1),\ldots,nat_{gt_k}(x_k))$$
$$st(p) = nat_{gt_1\underline{x}\ldots\underline{x}gt_k}(x_1 x \ldots x x_k)$$

(for x cartesian product of sets). Using the definition of $nat_{gt_1\underline{x}\ldots\underline{x}gt_k}$ in theorem 3.3:4 we get

$$nat_{gt_1\underline{x}\ldots\underline{x}gt_k} = (cross\cdot(nat_{gt_1}x\ \ldots))^X\cdot nat'(\underline{S}[\![gt_1]\!],\ldots)$$

Here nat' is given by theorem 3.2:16 so as for $take_i$ above

$$st(p) = (cross^{\cdot}(nat_{gt_1} \times \ldots))^{X'}(x_1 \times \ldots \times x_k)$$

where $\ldots^{X'}$ and cross' relate to the tensor product $\mathcal{P}(\underline{S}[\![gt_1]\!] \times \ldots)$.

But $cross'(x_1,\ldots,x_k) = x_1 \times \ldots \times x_k$ so $st(p) = ct(p)$ follows. ///

The interplay between S, C and sim

Before undertaking the structural induction that $sim_t()(\underline{S}[\![e]\!], \underline{C}[\![e]\!])$ holds it is convenient to consider the interplay between the functors $\underline{S}[\![t]\!]$ and $\underline{C}[\![t]\!]$ and the predicate sim_t. A similar situation arises in chapter 4 so to make the development immediately applicable there we consider arbitrary interpretations $\underline{I}$ and $\underline{J}$. For sim_t we assume that there are admissible predicates sm'_{ft} on $\underline{I}[\![ft]\!] \times \underline{J}[\![ft]\!]$ fulfilling that

$$sm'_{ft}(v,w) \text{ implies that } v=\perp \text{ if } w=\perp \qquad (*)$$

As has been said previously the need to assume (*) is because of the presence of the top-level domain constructor *.

Define the category $\underline{\underline{SIM}}$ as follows. The objects are triples (D,Q,L) where D and L are cpo's and Q is an admissible predicate on D×L that satisfies (*). A morphism from (D,Q,L) to (D',Q',L') is a pair (f,g) of $\underline{\underline{CPO2s}}$ morphisms $f:D \to D'$ and $g:L \to L'$ such that the following "naturality" conditions hold:

$$Q(d,l) \Rightarrow Q'(\ (f\downarrow 1)(d)\ ,\ (g\downarrow 1)(l)\ )$$

$$Q(\ (f\downarrow 2)(d')\ ,\ (g\downarrow 2)(l')\ ) \Leftarrow Q'(d',l')$$

Composition is defined componentwise and the identity on (D,Q,L) is (id_D, id_L) where id_D and id_L are the identities in $\underline{\underline{CPO2s}}$. It is straightforward to verify that this gives a category.

The notions of chain, cone, limiting cone and mediating morphism were explained in section 2.2 and also applies to $\underline{\underline{SIM}}$. The category $\underline{\underline{SIM}}$ admits the following analogue of theorem 2.2:2 and lemmas 2.2:5 and 2.2:6 for the characterisation of limiting cones. The result also holds for the variant of $\underline{\underline{SIM}}$ where the predicates are only required to be admissible.

Theorem 3.3:12. Consider a chain $\underline{E} = ((D_n,Q_n,L_n)_n,(e_n,f_n)_n)$ in $\underline{\underline{SIM}}$ such that e_n and f_n are embeddings of $\underline{\underline{CPO2s}}$ with upper adjoints e_n^R and f_n^R. A cone $\underline{R} = ((D,Q,L),(r_n,s_n)_n)$ such that r_n and s_n are embeddings of $\underline{\underline{CPO2s}}$ with upper adjoints r_n^R and s_n^R is limiting iff

$$id_D = \bigsqcup_n r_n \cdot r_n^R \quad \text{and} \quad id_L = \bigsqcup_n s_n \cdot s_n^R \quad \text{in } \underline{\underline{CPO2s}}$$

$$Q(d,l) \Xi \forall n: Q_n((r_n\lfloor 2)(d) , (s_n\lfloor 2)(l))$$

A limiting cone of this form always exists and the unique mediating morphism to another cone $\underline{R'} = ((D',Q',L'),(r_n',s_n')_n)$ is $(\bigsqcup_n r_n' \cdot r_n^R, \bigsqcup_n s_n' \cdot s_n^R)$. ///

Proof First note that $((D_n)_n,(e_n)_n)$ is a chain in $\underline{\underline{CPO2s}}$ for which lemma 2.2:5 is applicable. It follows that a cone $(D,(r_n)_n)$ with $r_n^U = r_n^R$ is limiting iff $id_D = \bigsqcup_n r_n \cdot r_n^R$. Further, such a limiting cone always exists and the unique mediating morphism to a cone $(D',(r_n')_n)$ is $\bigsqcup_n r_n' \cdot r_n^R$. Similarly for limiting cones $(L,(s_n)_n)$ of the chain $((L_n)_n,(f_n)_n)$.

We begin with proving the existence of a limiting cone $\underline{R}$ that satisfies the requirements. Let $(D,(r_n)_n)$ and $(L,(s_n)_n)$ be the limiting cones for $((D_n)_n,(e_n)_n)$ and $((L_n)_n,(f_n)_n)$ as indicated above. Define Q by the formula displayed in the theorem and note that this defines an admissible predicate that fulfils property (*). To see

that $((D,Q,L),(r_n,s_n)_n)$ is a cone reduces to showing

$$Q_n(d_n,l_n) \Rightarrow Q_m(\ (r_m\downarrow 2)((r_n\downarrow 1)(d_n))\ ,\ldots)$$

For $m \geq n$ we have $r_m\downarrow 2 \cdot r_n\downarrow 1 = e_{m-1}\downarrow 1 \cdot \ldots \cdot e_n\downarrow 1$ and for $m<n$ we have $r_m\downarrow 2 \cdot r_n\downarrow 1 = e_m\downarrow 2 \cdot \ldots \cdot e_{n-1}\downarrow 2$ as follows from consideration of the cone $(D,(r_n\downarrow 1)_n)$ for the chain $((D_n)_n,(e_n\downarrow 1)_n)$ in $\underline{\underline{\text{CPOs}}}$. Similarly for s_n and the result then follows because E is a chain in $\underline{\underline{\text{SIM}}}$.

We next show that the cone is limiting with the mediating morphism as stated in the theorem. Let $\underline{R}'$ be as displayed in the theorem and let (r,s) be the claimed mediating morphism. If (r',s') is another mediating morphism then in particular r' mediates from $(D,(r_n)_n)$ to $(D',(r'_n)_n)$ in $\underline{\underline{\text{CPO2s}}}$ and this shows that r=r'. Similarly s=s' so that (r,s) is the only candidate for the mediating morphism. To see it is the mediating morphism it suffices to show that (r,s) is a morphism of $\underline{\underline{\text{SIM}}}$ as the rest follows because r and s are mediating in $\underline{\underline{\text{CPO2s}}}$.

To see

$$Q(d,l) \Rightarrow Q'(\ (r\downarrow 1)(d)\ ,\ (s\downarrow 1)(l)\ )$$

note that Q(d,l) implies that

$$\forall n\colon Q_n(\ (r_n\downarrow 2)(d)\ ,\ (s_n\downarrow 2)(l)\ )$$

so by $\underline{R}'$ a cone

$$\forall n\colon Q'(\ (r'_n \cdot r_n^R)\downarrow 1(d)\ ,\ (s'_n \cdot s_n^R)\downarrow 1(l)\ )$$

and the admissibility of Q' then gives $Q'(\ (r\downarrow 1)(d)\ ,\ (s\downarrow 1)(l)\ )$.

To see

$$Q'(d',l') \Rightarrow Q(\ (r\downarrow 2)(d')\ ,\ (s\downarrow 2)(l')\ )$$

note that Q'(d',l') implies that

$$\forall n\colon Q_n(\ (r'_n\downarrow 2)(d')\ ,\ (s'_n\downarrow 2)(l')\ )$$

so by $\underline{R}$ a cone

$$\forall n\colon Q(((r_n' \cdot r_n^R)\downarrow 2)(d'),\ldots)$$

and the admissibility of Q then gives $Q(\ (r\downarrow 2)(d')\ ,\ (s\downarrow 2)(1')\ )$.

The statements of the theorem now holds for any limiting cone $\underline{R}'' = ((D'',Q'',L''),(r_n'',s_n'')_n)$ that satisfies $r_n^U = r_n^R$ and $s_n^U = s_n^R$. This is because any such cone is isomorphic to $\underline{R}$ that was explicitly constructed above (see section 2.2). For an example calculation let (r",s") be the isomorphism from $\underline{R}$ to $\underline{R}''$. Then Q"(d",1") iff $Q((r''\downarrow 2)(d''),\ldots)$ iff $\forall n\colon Q_n((r_n\downarrow 2)((r''\downarrow 2)(d')),\ldots)$ iff $\forall n\colon Q_n(\ (r_n''\downarrow 2)(d'')\ ,\ \ldots)$. ///

The main use of the category $\underline{\underline{SIM}}$ is as a convenient tool for expressing that $\underline{I}[\![t]\!]$, $\underline{J}[\![t]\!]$ and $sim_t[\![sm']\!]$ work together in a certain sense. This is formally expressed by proposing the following definition of a functor $\underline{IJ}[\![t]\!]$ over $\underline{\underline{SIM}}$. The effect on objects is

$$\underline{IJ}[\![t]\!]((D_1,Q_1,L_1),\ldots,(D_N,Q_N,L_N)) =$$
$$(\underline{I}[\![t]\!](D_1,\ldots,D_N)\ ,\ sim_t[\![sm']\!](Q_1,\ldots,Q_N)\ ,\ \underline{J}[\![t]\!](L_1,\ldots,L_N))$$

and the effect upon morphisms is

$$\underline{IJ}[\![t]\!]((f_1,g_1),\ldots,(f_N,g_N)) =$$
$$(\underline{I}[\![t]\!](f_1,\ldots,f_N)\ ,\ \underline{J}[\![t]\!](g_1,\ldots,g_N))$$

Theorem 3.3:13. $\underline{IJ}[\![t]\!]$ is a covariant functor over $\underline{\underline{SIM}}$. ///

Note If t does not mention the top-level smash product one could prove a similar theorem using the variant of $\underline{\underline{SIM}}$ where the predicates are only assumed to be admissible.

Proof By lemma 3.3:7 $\underline{IJ}[\![t]\!]((D_1,Q_1,L_1),\ldots)$ is an object of $\underline{\underline{SIM}}$. Assuming that $\underline{IJ}[\![t]\!]((f_1,g_1),..)$ is a morphism of $\underline{\underline{SIM}}$ it is immediate

to verify the functor laws using that $\underline{I}[\![t]\!]$ and $\underline{J}[\![t]\!]$ are functors and the componentwise definition of composition and the identities. Clearly $\underline{I}[\![t]\!](f_1,\ldots,f_N)$ and $\underline{J}[\![t]\!](g_1,\ldots,g_N)$ are morphisms of $\underline{\underline{\text{CPO2s}}}$ so to show that $IJ[\![t]\!]((f_1,g_1),\ldots)$ is a morphism of $\underline{\underline{\text{SIM}}}$ it suffices to prove the "naturality" conditions.

For this let (f_i,g_i) be $\underline{\underline{\text{SIM}}}$ morphisms with domain (D_i,Q_i,L_i) and range (D_i',Q_i',L_i'). Let (D,Q,L) be $\underline{IJ}[\![t]\!]((D_1,Q_1,L_1),\ldots)$ and let (D',Q',L') be defined similarly and let (f,g) be $\underline{IJ}[\![t]\!]((f_1,g_1),\ldots)$. It is to be shown that

$$Q(d,l) \Rightarrow Q'(\ (f\downarrow 1)(d)\ ,\ (g\downarrow 1)(l)\ )$$

$$Q(\ (f\downarrow 2)(d')\ ,\ (g\downarrow 2)(l')\ ) \Leftarrow Q'(d',l')$$

This is proved by structural induction on t.

<u>Case</u> $t=A_i$ is immediate because $f=id_D$ and $g=id_L$ (in $\underline{\underline{\text{CPO2s}}}$) and because Q equals Q'.

<u>Case</u> $t=t_1\times\ldots\times t_k$. Let d be an element of $\underline{I}[\![t]\!](D_1,\ldots)$, i.e. of $\underline{I}[\![t_1]\!](D_1,\ldots,D_N)\times\ldots\times\underline{I}[\![t_k]\!](D_1,\ldots,D_N)$ and similarly for l. If $Q(d,l)$ then for all i $sim_{t_i}[\![sm']\!](Q_1,\ldots,Q_N)(d\downarrow i,l\downarrow i)$. By the inductive hypothesis we have for all $i\leq k$ that

$$sim_{t_i}[\![sm']\!](Q_j',\ldots,Q_N')((\underline{I}[\![t_i]\!](f_1,\ldots,f_N)\downarrow 1)(d\downarrow i),\ \ldots)$$

so that $Q'(\ (f\downarrow 1)(d)\ ,\ (g\downarrow 1)(l)\ )$. The other implication is similar.

<u>Case</u> $t=t_1*\ldots*t_k$. As above $Q(d,l)$ gives $sim_{t_i}[\![sm']\!](Q_1',\ldots)(d_i',l_i')$ for $d_i'=(\underline{I}[\![t_i]\!](f_1,\ldots)\downarrow 1)(d\downarrow i)$ and l_i' defined similarly. If no d_i' or l_i' is $\perp$ then the result follows as before. Otherwise supppose without loss of generality that $d_i'=\perp$. By lemma 3.3:7 it follows that $l_i'=\perp$ so $(f\downarrow 1)(d)=\perp$ and $(g\downarrow 1)(l)=\perp$. Because $Q'(\perp,\perp)$ the result follows.

The other implication is similar.

Cases $t=t_1+\ldots+t_k$ and $t=t_{0\perp}$ are straightforward.

Case $t=t_1\rightarrow t_2$. Suppose $Q(d,1)$ and note that d is a continuous function from $\underline{I}[\![t_1]\!](D_1,\ldots)$ to $\underline{I}[\![t_2]\!](D_1,\ldots)$. Furthermore $(f\downarrow 1)(d)$ equals

$$(\underline{I}[\![t_2]\!](f_1,\ldots)\downarrow 1)\cdot d\cdot(\underline{I}[\![t_1]\!](f_1,\ldots)\downarrow 2)$$

(which will be abbreviated to $i_2\cdot d\cdot i_1$) and is a continuous function from $\underline{I}[\![t_1]\!](D_1',\ldots)$ to $\underline{I}[\![t_2]\!](D_1',\ldots)$. Similarly for $(g\downarrow 1)(1)$ and to show $Q'((f\downarrow 1)(d),(g\downarrow 1)(1))$ we must consider (v,w) such that $\mathrm{sim}_{t_1}[\![sm']\!](Q_1',\ldots,Q_N')(v,w)$ holds and show that

$$\mathrm{sim}_{t_2}[\![sm']\!](Q_1',\ldots,Q_N')((f\downarrow 1)(d)(v),(g\downarrow 1)(d)(w))$$

holds. By the inductive hypothesis

$$\mathrm{sim}_{t_1}[\![sm']\!](Q_1,\ldots,Q_N)(i_1(v),\ldots)$$

so that $Q(d,1)$ gives

$$\mathrm{sim}_{t_2}[\![sm']\!](Q_1,\ldots,Q_N)(d(i_1(v)),\ldots)$$

and by the inductive hypothesis the desired result follows. (Note that this proof used that the "naturality" conditions express the effect of "going in both directions" because of the contravariance of the function space construction in $\underline{\underline{\mathrm{CPO}}}$.) The other implication is similar.

Case $t=\mathrm{rec}X.t_0$. It is convenient to assume $V\vdash t$ for $\mathrm{card}(V)=N$ and that X has index $N+1$ in $V\cup\{X\}$. (The proof can easily be rectified if this is not the case.) Define the following functors over $\underline{\underline{\underline{\underline{\underline{\mathrm{CPO2s}}}}}}$

$$F_0=K_U,\quad F_{n+1}=\underline{I}[\![t_0]\!]\cdot(P_1,\ldots,P_N,F_n)$$
$$G_0=K_U,\quad G_{n+1}=\underline{J}[\![t_0]\!]\cdot(P_1,\ldots,P_N,G_n)$$

Let the limiting cone of the CPO2s chain

$$((F_n(D_1,\ldots,D_N)_n,(F_n(id_{D_1},\ldots,id_{D_N}))_n)$$

be $(D,(r_n)_n)$ and recall that $r_n{}^U=r_n{}^R$. Similarly $(D',(r_n')_n)$, $(L,(s_n)_n)$ and $(L',(s_n')_n)$ are defined as limits for the obviously intended chains. Define predicates R_n on $F_n(D_1,\ldots,D_N)\times G_n(L_1,\ldots,L_N)$ and similarly R_n' by the formulae

$$R_0(v,w)\Xi tt; \quad R_{n+1}=sim_{t_0}\lceil sm'\rceil(Q_1,\ldots,Q_N,R_n)$$
$$R_0'(v',w')\Xi tt, \quad R_{n+1}'=sim_{t_0}\lceil sm'\rceil(Q_1',\ldots,Q_N',R_n')$$

It is an easy numerical induction that

$$(F_n(f_1,\ldots,f_N),G_n(g_1,\ldots,g_N))$$

is a SIM morphism from

$$(F_n(D_1,\ldots,D_N)\ ,\ R_n\ ,\ G_n(L_1,\ldots,L_N))$$

to

$$(F_n(D_1',\ldots,D_N'))\ ,\ R_n'\ ,\ G_n(L_1',\ldots,L_N'))$$

To prove the "naturality" condition that Q(d,l) implies $Q'((f\downarrow 1)(d),(g\downarrow 1)(l))$ suppose Q(d,l). By definition of Q

$$\forall n:\ R_n((r_n\downarrow 2)(d),(s_n\downarrow 2)(l))$$

so because $(F_n(f_1,\ldots,f_N),G_n(g_1,\ldots,g_N))$ is a morphism

$$\forall n:\ R_n'(\ (F_n(f_1,\ldots,f_N)\downarrow 1)((r_n\downarrow 2)(d))\ ,\ \ldots)$$

By lemma 2.2:11 (second half) and the definition of f

$$\forall n:\ R_n'(\ (r_n'\downarrow 2)((f\downarrow 1)(d))\ ,\ \ldots)$$

so that by definition of R the result follows. The other "naturality" is similar.

Cases t=X and t=gt→gt' are straightforward. ///

This theorem may be compared with lemma 13.65 in /Sto77/ and the results in /Rey74/ about how to obtain a so-called relational functor. The local continuity of $\underline{I}[\![t]\!]$ and $\underline{J}[\![t]\!]$ ensure that the similar condition holds for $\underline{IJ}[\![t]\!]$. This is sufficient for obtaining a continuity result analogous to lemma 2.2:3, namely that $\underline{IJ}[\![t]\!]$ preserves limiting cones for chains like those mentioned in theorem 3.3:12. (The details may be extracted from the proof of the following theorem.)

...

..

The connection between $S[\![e]\!]$ and $C[\![e]\!]$

We are now ready to perform the structural induction that shows $sim_t()(\underline{S}[\![e]\!],\underline{C}[\![e]\!])$ and thereby establishes the relationship between $\underline{S}$ and $\underline{C}$.

Theorem 3.3:14.

(1) If $\emptyset \vdash t : e$ then $sim_t()(\underline{S}[\![e]\!],\underline{C}[\![e]\!])$

(2) In general consider interpretations $\underline{I}$ and $\underline{J}$ and admissible predicates sm'_{ft} on $\underline{I}[\![ft]\!]\times\underline{J}[\![ft]\!]$. Suppose $\emptyset \vdash e:t$ and that an analogue of lemma 3.3:11 holds for $\underline{I}$, $\underline{J}$ and $sim_t[sm']$. Then $sim_t()(\underline{I}[\![e]\!],\underline{J}[\![e]\!])$ holds if

(a) t contains no top-level smash product

or if each sm'_{ft} satisfies that

(b) $sm'_{ft}(v,w)$ implies that $v=\perp$ iff $w=\perp$ (*) ///

Proof Clearly (1) is a special case of (2b). The proof of (2a) is similar to the proof of (2b) but uses the variant of $\underline{\underline{SIM}}$ where predicates are only assumed to be admissible. To prove (2b) the following more general result is proved by structural induction on e:

if $tenv \vdash e:t$ and for all x: $sim_{tenv(x)}[sm']()(ienv(x),jenv(x))$

then $sim_t[sm']()(\underline{I}[\![e]\!](ienv),\underline{J}[\![e]\!](jenv))$

Most cases are straightforward using that an analogue of 3.3:11 holds. The cases considered below are among the harder ones.

Case $e = (*e_1,\ldots,e_k*)$. The inductive hypothesis gives

$$sim_{t_i}[sm'](\,)(\underline{I}[\![e_i]\!](ienv),\underline{J}[\![e_i]\!](jenv))$$

so that

$$sim_t[sm'](\,)(\underline{I}[\![e]\!](ienv),\underline{J}[\![e]\!](jenv))$$

is immediate in the case where no $\underline{I}[\![e_i]\!](ienv)$ or $\underline{J}[\![e_i]\!](jenv)$ is $\perp$. If one is suppose it is $\underline{I}[\![e_i]\!](ienv)$. Lemma 3.3:7 shows that $\underline{J}[\![e_i]\!](jenv)$ also is so the result follows because $sim_t[sm'](\,)(\perp,\perp)$.

Case $e = Y\ e'$. Here $\underline{I}[\![e]\!](ienv) = \bigsqcup_n (\underline{I}[\![e']\!](ienv))^n(\perp)$ and similarly for $\underline{J}$. Clearly $sim_t[sm'](\,)(\perp,\perp)$ holds so by the inductive hypothesis

$$sim_t[sm'](\,)\ (\ (\underline{I}[\![e']\!](ienv))^n(\perp)\ ,\ \ldots\ )$$

and admissibility then gives $sim_t[sm'](\,)(\underline{I}[\![e]\!](ienv)\ ,\ \ldots\ )$.

Cases e=mkrec e' and e=unrec e". Let t_0 be given such that $tenv \vdash e' : t_0[recX.t_0/X]$ and $tenv \vdash e" : recX.t_0$ and note that $\{X\} \vdash t_0$. Write $D=\underline{I}[\![recX.t_0]\!]$ and $L=\underline{J}[\![recX.t_0]\!]$ and let $i:\underline{I}[\![t_0]\!](D)\to D$ and $j:\underline{J}[\![t_0]\!](L)\to L$ be the isomorphisms of CPOs such that (by lemma 2.3:2)

$$\underline{I}[\![mkrec\ e']\!](ienv) = i(\underline{I}[\![e']\!](ienv))$$

$$\underline{I}[\![unrec\ e"]\!](ienv) = i^{-1}(\underline{I}[\![e"]\!](ienv))$$

and similarly for j. The result is straightforward from the inductive hypothesis and lemma 3.3:8 once it has been shown that

$$sim_{recX.t_0}[sm'](\,)(i(v),j(w))$$
$$\text{iff } sim_{t_0}[sm'](sim_{recX.t_0}[sm'](\,))(v,w)$$

To show this we consider $\underline{IJ}[\![t_0]\!]$ over $\underline{\underline{SIM}}$. Define the predicate true by $true(v,w) \equiv tt$. Using theorem 3.3:13 it is easy to see that

$$((\underline{IJ}[\![t_0]\!]^n(U,true,U))_n \ , \ (\underline{IJ}[\![t_0]\!]^n \ (\bot,\bot))_n)$$

is a chain in $\underline{\underline{SIM}}$. The limiting cone is

$$((D,Q,L),(r_n,s_n)_n)$$

for $Q=sim_{recX.t_0[\![sm']\!]}()$ and r_n, s_n as in theorem 3.3:12. It follows from section 2.2 that $(D,(r_n)_n)$ is the limiting cone in $\underline{\underline{CPO2s}}$ of the chain $((\underline{I}[\![t_0]\!]^n(U))_n,(\underline{I}[\![t_0]\!]^n(\bot))_n)$ and similarly for $(L,(s_n)_n)$. Next define $r'_0=\bot$ and $r'_{n+1}=\underline{I}[\![t_0]\!](r_n)$ and similarly for s'_n. From theorem 3.3:13 it follows that

$$\underline{R} = (\ (\underline{I}[\![t_0]\!](D),sim_{t_0[\![sm']\!](Q)},\underline{J}[\![t_0]\!](L)) \ , \ (r'_n,s'_n)_n \ )$$

is also a cone. Therefore there is a unique mediating morphism (r,s) from the limiting cone to $\underline{R}$. Theorem 3.3:12 and lemma 2.2:5 show that r is the unique mediating morphism in $\underline{\underline{CPO2s}}$ from $(D,(r_n)_n)$ to $(\underline{I}[\![t_0]\!](D),(r'_n)_n)$. Since $\underline{I}[\![t_0]\!]$ is locally continuous r is an isomorphism and in fact $r = (i^{-1},i)$. Similarly $s = (j^{-1},j)$ and the desired result follows because (r,s) is a morphism of $\underline{\underline{SIM}}$. (This argument is easily extended to show that $\underline{R}$ is also a limiting cone and thereby proves the continuity result for $\underline{IJ}[\![t]\!]$ that was mentioned above.) ///

This theorem may be compared to /Nie82/ where a similar theorem (but with program points) is proved for a simple imperative language. The theorem there is based on a store semantics /MiSt76/ rather than a standard semantics and it is said (p.274 lines 1 to 4) that the theorem would not hold for the obvious standard semantics. The corresponding remark here is that it is essential to use the tensor

product for $\underline{C}(\underline{x})$ rather than the cartesian product. For use of the cartesian product might suggest

$$\underline{C}(\text{tuple})(g_1,g_2) = \lambda 1.(g_1(1),g_2(1))$$

$$\underline{C}(\text{add}) = \lambda(Y,Z).\ \{y+z \mid y\in Y \wedge z\in Z\}_R$$

(assuming that the natural equivalences are the identities). Then $\underline{C}[\![\text{add}\circ\text{tuple}(\text{id},\text{id})]\!]\{2,3\}_R$ gives $\{4,5,6\}_R$ instead of the correct $\{4,6\}_R$. This corresponds to what is done in /Don78/. The problem is that the connection between the pairs of values is lost. A similar problem is likely to occur if t::=gt is allowed (e.g. $\underline{C}[\![\lambda x.\text{add}(x,x)]\!]$).

Remark We conclude this section with a remark about the definition of $\text{sim}_{\text{rec}X.t}()(d,l)$. Its definition was of the form

$$\forall n:\ \text{SIM}_{t,n}(\ r_n^U(d)\ ,\ s_n^U(l)\ ) \qquad (1)$$

for embeddings r_n and s_n of $\underline{\underline{\text{CPOs}}}$. The definition of REC in section 2.2 was also given using embeddings but it was shown that a least fixed point definition could be used instead. Similar remarks apply to $\text{view}_{\text{rec}X.t}$. When predicates are ordered by "is implied by" the analogue of a least fixed point definition is

$$\forall n:\ Q_n(d,l) \qquad (2)$$

where $Q_0(d,l)$ is always true and

$$Q_{n+1}(d,l) \equiv \text{sim}_t(Q_n)(r^{-1}(d),s^{-1}(l))$$

for isomorphisms r and s in $\underline{\underline{\text{CPOs}}}$.

Theorem 3.3:13 and induction can be used to relate $\text{SIM}_{t,n}$ and Q_n by showing that

$$((r_n,r_n^U),(s_n,s_n^U)) : (\underline{S}[\![t]\!]^n(U)\ ,\ \text{SIM}_{t,n}\ ,\ \underline{C}[\![t]\!]^n(U)) \rightarrow (\underline{S}[\![\text{rec}X.t]\!]\ ,\ Q_n\ ,\ \underline{C}[\![\text{rec}X.t]\!])$$

is a morphism of $\underline{\underline{SIM}}$. It follows that (1) is equivalent to

$$\forall n: Q_n(r_n(r_n^U(d)) , s_n(s_n^U(1))) \qquad (3)$$

and that (2) implies (1). The converse implication may fail as will be sketched below. Intuitively this should not be surprising because the definition of $sim_{t_1 \to t_2}$ is such that $sim_t(Q)$ does not necessarily depend monotonically on Q.

For a sketch of an example where (1) does not imply (2) consider

$$t = (U_\perp + N_\perp + X) + N_\perp + (X \to X)$$

Intuitively $\underline{S}[\![recX.t]\!]$ can code (among other things) lists of integers, integers and functions from lists of integers to integers. Let d be the coding of the function that maps the coding of the list $(i,n_1,\ldots,n_k)$ to the coding of n_i if i is between 1 and k and gives the coding of 1 otherwise. Since $\underline{C}[\![recX.t]\!] = \underline{S}[\![recX.t]\!]$ we may take 1 = d and (1) is straightforward. On the other hand (2) fails because already $Q_2(d,1)$ fails. The codings of the lists (1,2) and (1,3) are related by Q_1 but this is not the case for the codings of the results (2 and 3). The problem is that d and 1 are not restricted in the number of unfoldings they are allowed to perform.

The failure of the attempt (2) at defining $sim_{recX.t}()$ as a least fixed point may be circumvented by placing suitable restrictions on recX.t. However, even in the general case $sim_{recX,t}()$ is essentially the only solution to the following equation

$$Q \equiv \left(\forall d,1: Q(d,1) \Leftrightarrow sim_t(Q)(r^{-1}(d),s^{-1}(1))\right)$$

To be precise: if an admissible predicate Q is a solution and Q(d,1) implies that $d=\perp$ iff $1=\perp$ then Q equals $sim_{recX.t}()$. It follows from the proof of theorem 3.3:14 that $sim_{recX.t}()$ is a solution. Next if

Q is an solution it follows that

$$((r,r^{-1}),(s,s^{-1})):$$

$$(\underline{S}[\![t[recX.t/X]]\!],\ sim_t(Q),\ \underline{C}[\![t[recX.t/X]]\!]) \rightarrow$$

$$(\underline{S}[\![recX.t]\!],\ Q,\ \underline{C}[\![recX.t]\!])$$

is a morphism of $\underline{\underline{SIM}}$. One can then construct a cone

$$\underline{R} = ((\underline{S}[\![recX.t]\!],\ Q,\ \underline{C}[\![recX.t]\!]),\ \ldots)$$

$$\ldots$$

for the chain

$$((\underline{SC}[\![t]\!]^n(U))_n\ ,\ (\underline{SC}[\![t]\!]^n(\perp))_n)$$

Since the mediating morphism from the limiting cone to $\underline{R}$ is (id,id) the result follows. ///

4 ABSTRACT INTERPRETATION

In this chapter abstract interpretation is developed for the meta-language. This builds on a study of the relation "is a safe approximation to" between (forward) data flow analyses specified as interpretations. As in chapter 1 this is done using pairs of abstraction and concretization functions. Conditions upon such pairs are studied in section 4.1.

The overall framework is developed in section 4.2. In this development certain problems arise because of the presence of the top-level smash product which therefore needs to be abandoned in some parts of the development. Using a family $(con_{gt})_{gt}$ of concretization functions a relation is defined between two interpretations (say $\underline{I}$ and $\underline{J}$) that specify data flow analyses. The relation is sufficient and (essentially) necessary for

$$\underline{I}[\![e]\!]\cdot con_{gt} \sqsubseteq con_{gt'}\cdot \underline{J}[\![e]\!]$$

to hold whenever $\emptyset \vdash e:gt \rightarrow gt'$. Given $\underline{I}$ and additionally a family $(abs_{gt})_{gt}$ of abstraction functions one can define an interpretation $\underline{I}'$ that gives as faithful a picture of $\underline{I}$ as is possible using the (generally more approximate) spaces of $\underline{I}'$. The interpretation $\underline{I}'$ is said to be induced from $\underline{I}$ and is a convenient way of specifying data flow analyses (not least if $\underline{I}$ is the collecting interpretation $\underline{C}$).

In the above it was not required that con_{gt} was defined structurally on gt. In keeping with the principle of compositionality of

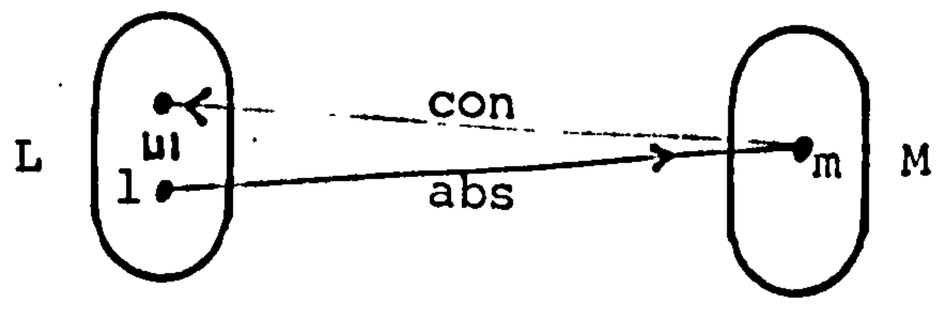
denotational semantics we study in section 4.3 how a compositional definition can be given. In part this resembles the construction of the natural equivalence in the proof of theorem 3.3:4. In section 4.4 the induced functionals are studied. Taking composition ◻ as an example one might expect it to be functional composition. Similarly for other functionals expected formulations may be given. Because "inducing" is very precise the induced version of an expected definition need not, however, itself be an expected definition. It is shown that under certain conditions it is safe to replace the induced functionals with the expected definitions. Finally, section 4.5 discusses the role of the tensor product in generalising the relational flow analysis method.

4.1 PAIRS OF ADJOINED FUNCTIONS

In the introduction we considered approximation between complete lattices. The partial orders were viewed as representing the notion of "can safely be approximated by". To describe approximation a concretization function con and an abstraction function abs were used. It was argued that they must satisfy that

$$abs(l) \sqsubseteq m \quad \text{iff} \quad l \sqsubseteq con(m)$$

This is equivalent to assuming abs and con to be monotonic and adjoined: $con \circ abs \sqsupseteq id$ as is illustrated by



and $abs \circ con \sqsubseteq id$. If additionally $abs \circ con = id$ we shall say that abs and con are exactly adjoined /Nie81a/. This has often been assumed

(e.g. /CoCo77b/) because then con is one-one and M contains no superfluous elements. Part of the development in this chapter can be performed using weaker assumptions than adjoined (see /Nie82/) but this will not be considered here.

These concepts can be defined relative to any cpo-category. A pair $(f:L\to M, g:M\to L)$ of morphisms is said to be <u>adjoined</u> iff $f\cdot g\sqsubseteq id$ and $g\cdot f\sqsupseteq id$. A morphism f is a <u>lower adjoint</u> iff there exists a morphism g such that (f,g) is an adjoined pair. The g need not exist but if it does it is uniquely determined: for if both g and g' would do then

$$g = g\cdot id \sqsupseteq g\cdot(f\cdot g') = (g\cdot f)\cdot g' \sqsupseteq id\cdot g' = g'$$

and similarly $g'\sqsubseteq g$. A morphism g is an <u>upper adjoint</u> if there exists a (necessarily unique) morphism f such that (f,g) is an adjoined pair. An adjoined pair (f,g) with $f\cdot g=id$ is said to be <u>exactly adjoined</u>.

It will be seen that in the previous paragraph a category $\underline{\underline{\underline{CLm}}}$ of complete lattices and monotonic functions was presupposed.

In most of this chapter we shall assume pairs (abs,con) of abstraction and concretization functions to be adjoined relative to $\underline{\underline{\underline{\underline{ACLs}}}}$. The use of this category will be discussed below. It is straightforward to show for such a pair of adjoined functions that the lower adjoint abs is completely additive and the upper adjoint con is completely multiplicative (the dual notion of preserving greatest lower bounds). Further the formulae /CoCo79/

$$abs(l) = \bigsqcap\{m \mid l\sqsubseteq con(m)\}$$

$$con(m) = \bigsqcup\{l \mid abs(l)\sqsubseteq m\}$$

are immediate.

Restrictions due to ACLs

We now argue that it is not overly restrictive to assume that all pairs (abs,con) of abstraction and concretization functions are adjoined relative to ACLs. The restrictions must be compared with those that are commonly made in abstract interpretation and monotone frameworks.

Abstract interpretation (as explained in the Introduction and e.g. /CoCo77b/) may be viewed as using CLm instead of ACLs. The use of ACLs means that (precisely) two additional restrictions are imposed. One is that the objects (complete lattices) must be algebraic. The other is that the (monotonic) concretization functions must be strict and continuous.

The formulation of monotone frameworks /KaUl77/ considers objects that are semi-lattices of finite height that have a least element. Such objects are complete lattices and they are algebraic iff they are countable (as will be the case in applications). Functions upon such objects are assumed to be monotonic but because of the finite height this is equivalent to assuming that they are continuous.

The assumption that concretization functions be continuous is motivated by the desire to stay within the ususal "continuous" domain theory. It is a restriction with respect to abstract interpretation but hardly in practice, i.e. not with respect to monotone frameworks. The assumption of strictness is motivated by the presence of domain constructors (like $\otimes$) that are only functors over a category of strict functions. This should be no serious restriction as an adjoined situation

$$(abs: L \to M,\ con: M \to L)$$

may be replaced by

$$(\text{strict}(\text{up}^{\circ}\text{abs}):L\to M_{\perp},\ \text{lift}(\text{con}):M_{\perp}\to L)$$

and here lift(con) is strict.

Algebraicity of objects is motivated by the construction of the tensor product in section 3.2. To see this is not overly restrictive consider first objects corresponding to bottom-level types in the collecting interpretation $\underset{\sim}{C}$. These are algebraic complete lattices because they are (isomorphic to) the powerdomain of the corresponding objects in the standard interpretation. Consider next the corresponding object in an interpretation that specifies a data flow analysis. If it is related to the object in $\underline{C}$ by a pair of exactly adjoined functions then it will be algebraic too. This is a consequence of:

Lemma 4.1:1. Let L be an algebraic complete lattice and M a partially ordered set. Let $abs:L\to M$ and $con:M\to L$ be monotonic functions such that

$$con^{\circ}abs \sqsupseteq id \quad \text{and} \quad abs^{\circ}con = id$$

Then M is a complete lattice with least upper bound of a subset X given by $abs(\bigsqcup\{con(m) \mid m\in X\})$. If additionally con is continuous then M is algebraic with $B_M = \{abs(l) \mid l\in B_L\}$. ///

Proof For the first result we calculate

$$\begin{array}{l} \forall m\in X:\ m\sqsubseteq m' \\ \Updownarrow \\ \forall m\in X:\ con(m)\sqsubseteq con(m') \\ \Updownarrow \\ \bigsqcup\{con(m) \mid m\in X\} \sqsubseteq con(m') \\ \Updownarrow \\ abs(\bigsqcup\{con(m) \mid m\in X\}) \sqsubseteq m' \end{array}$$

and this shows that M is a complete lattice with least upper bounds

as stated. For the second result let $b \in B_L$ and show $abs(b) \in B_M$. To see this let $abs(b) \sqsubseteq \bigsqcup_i m_i$ so that $b \sqsubseteq con(\bigsqcup_i m_i) = \bigsqcup_i con(m_i)$. Then there is i such that $b \sqsubseteq con(m_i)$ hence $abs(b) \sqsubseteq m_i$. To see that M is algebraic with B_M as stated we use fact 2.2:1 so it suffices to note for $m \in M$ that

$$m = abs(con(m)) = abs(\bigsqcup\{b \in B_L \mid b \sqsubseteq con(m)\})$$
$$= \bigsqcup\{abs(b) \mid b \in B_L \wedge b \sqsubseteq con(m)\}$$
$$= \bigsqcup\{abs(b) \mid b \in B_L \wedge abs(b) \sqsubseteq m\}$$

and that $\{abs(b) \mid b \in B_L \wedge abs(b) \sqsubseteq m\}$ is directed. ///

The lemma may fail if abs·con=id was weakened to abs·con$\sqsubseteq$id. As an example let M be a cpo that is not algebraic and not a complete lattice. Taking L=U, abs=$\perp$, con=$\perp$ the assumptions of the lemma are fulfilled. If con is continuous, however, we have that $\{abs(b) \mid b \in B_L\}$ is a subset of B_M.

Taking it for granted that objects should be algebraic one can give the following intuitive argument for why they should be complete lattices. First they should be cpo's if iteration and recursion (Y in the metalanguage) is still to be treated by a least fixed point approach as is also the case in the MFP solution of data flow analysis. Secondly objects should be semi-lattices so that the effect of conditional (cond in the metalanguage) may be expressed using a least upper bound as in $\underline{C}$(cond). Lemma 3.2:2 then shows that objects should be complete lattices.

Considering further restrictions

We now concentrate on a pair (abs:L→M,con:M→L) of abstraction and concretization functions that is adjoined relative to ACLs. The abstraction function is strict, continuous and additive (hence completely addditive) and the concretization function is strict and continuous but not additive. It follows from the discussion after the previous lemma that abs sends a finite element to a finite element. This is equivalent to saying that it specializes to a continuous function $abs:B_L \rightarrow B_M$. We now investigate the cases where it specializes to $abs:PB_L \rightarrow PB_M$ or $abs:IB_L \rightarrow IB_M$. (Recall that PB is the set of finite elements that are primes and similarly IB for irreducible.)

Preservation of primes is particularly of interest for powerdomains.

Lemma 4.1:2. Let (abs:L→M,con:M→L) be a pair of adjoined functions where L is isomorphic to a powerdomain. Then abs specializes to $abs:PB_L \rightarrow PB_M$ iff con is additive. ///

Proof For "if" suppose that con is additive and that b is a finite prime of L. Clearly abs(b) is finite and to see that it is a prime suppose that $abs(b) \sqsubseteq m_1 \sqcup m_2$. Then $b \sqsubseteq con(m_1) \sqcup con(m_2)$ so for some i we have $b \sqsubseteq con(m_i)$ and hence $abs(b) \sqsubseteq m_i$. For "only if" suppose that abs specializes to $abs:PB_L \rightarrow PB_M$. Then the calculation

$$con(m_1 \sqcup m_2) =$$

(algebraicity of L and theorem 3.1:5)

$$\bigsqcup\{b \in PB_L \mid b \sqsubseteq con(m_1 \sqcup m_2)\} =$$

$$\bigsqcup\{b \in PB_L \mid abs(b) \sqsubseteq m_1 \sqcup m_2\} =$$

$$con(m_1) \sqcup con(m_2)$$

shows that con is additive. ///

"If" holds even if L is not isomorphic to a powerdomain but "only if" may fail if this is not the case.

Suppose now that $L=\mathcal{P}(D)$ and $M=\mathcal{P}(E)$. If abs specializes to $abs:PB_L \to PB_M$ it is possible to define a representation function $r:D \to E$ by

$$abs(\{d\}_R) = \{r(d)\}_R$$

This defines a continuous function r that is strongly strict, i.e. $r(d)=\perp$ iff $d=\perp$, and it specializes to $r:B_D \to B_E$. It is immediate to verify the formulae

$$abs(I) = LC(\{r(b) \mid b \in I\})$$

$$con(I') = \{b \in B_D \mid r(b) \in I'\}$$

Conversely for any strongly strict and continuous function $r:B_D \to B_E$ the above formulae define a pair of adjoined functions with con additive. An example representation function is $r:Z_\perp \to \{-,0,+\}_\perp$ defined by r(-1)=- etc. (as was used in the Introduction). It will, however, be too restrictive to assume all concretization functions to be additive as this will make it impossible to handle many data flow analyses that can otherwise be handled. Examples include constant propagation and the example in the Introduction where an independent attribute method is related to a relational method.

The situation where abs specializes to a map $abs:IB_L \to IB_M$ will be found of use in section 4.4. We begin with motivating the intuitive role played by the irreducible finite elements. An element l of L is said to be <u>essential</u> iff whenever $l=\bigsqcup X$ for a subset X of L then l is an element of X or $l=\perp$. Intuitively l can only be

specified by itself. There is an immediate analogy with the definition of finite element in section 2.2; there X was assumed to be a countable and directed subset and $1=\bigsqcup X$ was weakened to $1\sqsubseteq\bigsqcup X$.

Lemma 4.1:3. IB_L is the set of finite elements that are essential.///

Proof An essential element is clearly irreducible so a finite and essential element is in IB_L. Conversely let i be an element of IB_L and X a subset of L such that $i=\bigsqcup X$. The algebraicity of L gives $\bigsqcup X = \bigsqcup LC(X)$ for LC as defined in section 3.1. If LC(X) is empty we have $i=\bot$ which is essential so assume otherwise. Then there is a total and onto map $k:N\rightarrow LC(X)$ and the axiom of choice guarantees a total map $k':N\rightarrow X$ such that $k(n)\sqsubseteq k'(n)$. It follows that $\bigsqcup X = \bigsqcup\{k'(n) \mid n\in N\}$. Now $(\bigsqcup_{n=1}^{m} k'(n))_m$ is a chain and $i\sqsubseteq\bigsqcup X$ gives $i\sqsubseteq\bigsqcup_{n=1}^{m} k'(n)$ for some m (because i is finite). Clearly $i=\bigsqcup X$ gives $i = k'(1)\sqcup\ldots\sqcup k'(m)$ so because i is irreducible there is m' such that $i = k'(m')$. It follows that i is an element of X. ///

That abs specializes to $abs:IB_L\rightarrow IB_M$ therefore says that abstracting an essential piece of information must give an essential piece of information. It is a restriction to assume abstraction functions to satisfy this but we shall argue that it may be acceptable to do so. For this we consider conditions upon the upper adjoint $con:M\rightarrow L$. It is said to be "<u>irreducibly covered</u>" iff

$$\forall m\in M:\ \forall i\in IB_L:\ i\sqsubseteq con(m) \Rightarrow \exists j\in IB_M:\ j\sqsubseteq m \wedge i\sqsubseteq con(j)$$

and "<u>irreducibly generated</u>" iff con=lin(con), i.e. iff

$$\forall m\in M:\ con(m) = \bigsqcup\{con(j) \mid j\in IB_M \wedge j\sqsubseteq m\}$$

Lemma 4.1:4. For a pair (abs:L→M, con:M→L) of adjoined functions the statements

(1) abs specializes to abs:$IB_L \to IB_M$

(2) con is irreducibly covered

are equivalent and if L is isomorphic to a powerdomain they are equivalent to

(3) con is irreducibly generated ///

Proof If (1) holds then (2) it shown by "choosing" j = abs(i) in the definition of irreducibly covered. If (2) holds then (1) is shown by instantiating the definition of irredicibly covered with $i \in IB_L$ and m=abs(i). Assume now that L is isomorphic to a powerdomain. If (3) holds then to show (1) consider $i \in IB_L$ and calculate

$$i \sqsubseteq con(abs(i)) = \bigsqcup \{con(j) \mid j \in IB_M \wedge j \sqsubseteq abs(i)\}$$

But, by theorem 3.1:7, $i \in PB_L$ and therefore $i \sqsubseteq con(j)$ for some $j \in IB_M$ with $j \sqsubseteq abs(i)$. This shows abs(i)=j and hence $abs(i) \in IB_M$. Finally, if (1) holds then (3) is proved as follows. First

$$con(m) \sqsupseteq \bigsqcup \{con(j) \mid j \in IB_M \wedge j \sqsubseteq m\}$$

is immediate. For the other inclusion note that (by theorems 3.1:5 and 3.1:7)

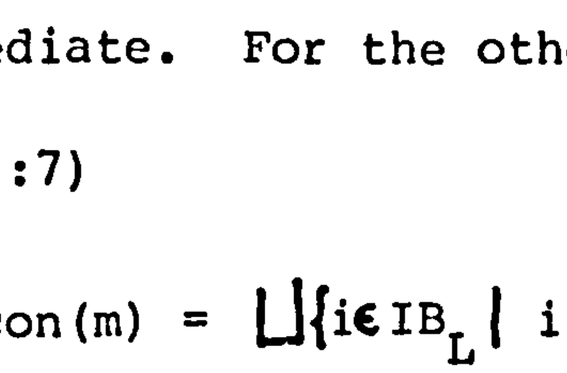
$$con(m) = \bigsqcup \{i \in IB_L \mid i \sqsubseteq con(m)\}$$

so that one may calculate

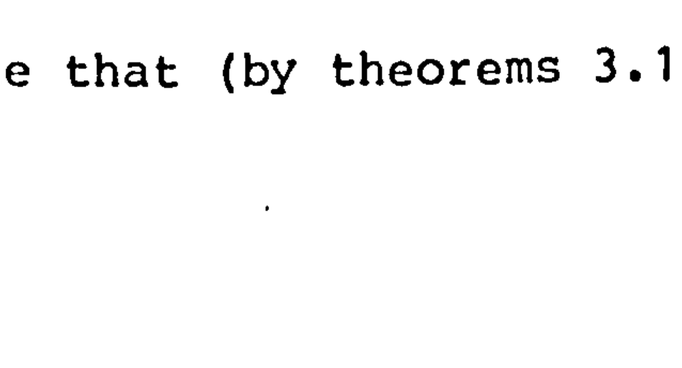
$$con(m) \sqsubseteq \bigsqcup \{con(abs(i)) \mid i \in IB_L \wedge abs(i) \sqsubseteq m\}$$
$$\sqsubseteq \bigsqcup \{con(j) \mid j \in IB_M \wedge j \sqsubseteq m\}$$ ///

To illustrate these conditions choose $L=\wp(N_\perp)$ and

M_1 =



M_2 =



Define pairs of adjoined functions (abs_i, con_i) by $con_2(large)=\{3,4,...\}_\perp$ and $con_i(2)=\{2\}_R$ etc. Then con_2 but not con_1 is irreducibly generated. Examples like con_1 do come up occasionally /Jon81a/ so it is a restriction to require concretization functions to be irreducibly generated. It is the author's point of view that this restriction is acceptable. This is partly because the example in /Jon81a/ has always been "somewhat unnatural" in the view of the author and partly because the restriction is motivated by the consideration of essential elements. At any rate the restriction will be of use in a small part of the development in section 4.4. When L is not a powerdomain the consideration of essential elements suggests that "irreducibly covered" rather than "irreducibly generated" is the condition to be imposed upon concretization functions. This condition is preserved by composition of upper adjoints because (1) in the theorem is preserved when composing the corresponding lower adjoints. (The second condition is not preserved under composition: consider $f:M_1\to M_2$ and $g:M_2\to\{T\}_\perp$ defined by $f(x)=x$ and $g(x)=(x\stackrel{?}{=}large)\to T,\perp)$

It is appropriate at this point to give a comparison with /WaSh77/ where the semantics of data types is studied. (In giving the comparison some technical differences will be ignored.) Starting with an algebraic cpo D they give axioms for a "type structure" M over D. The axioms prescribe that the elements of M must be elements of $\mathcal{P}(D)$ and M is partially ordered in the same way that $\mathcal{P}(D)$ is, i.e. by subset inclusion. Further, M is a complete lattice and it is possible that $PB_M=\{\perp\}$. What is interesting is that IB_M is isomorphic to B_D. This means that their notion of the "tightening" of h is nothing but lin(h). Further they use "tightening" to prove certain results by "case analysis" where a "case" is an element of

IB_M. This idea is intuitively close to the arguments to be given in section 4.4 for the use of lin in the expected definitions for $filter_x$ and $take_i$ (for tensor products).

4.2 APPROXIMATING INTERPRETATIONS

In this section we define a relation between those interpretations that specify data flow analyses. The relation guarantees that the semantics given to certain expressions by one interpretation is safely approximated by the other. For specifying data flow analyses the immediate possibility is to specify an interpretation in full. We explore another possibility namely to induce an interpretation, i.e. to define an interpretation that mimicks a given interpretation but using more approximate spaces.

For this development it is helpful to have additional assumptions about the interpretations that specify data flow analyses. Define an approximating interpretation to be an interpretation $\underline{I}$ such that:

- the categories $\underline{I}(\underline{\underline{B}})$ and $\underline{I}(\underline{\underline{Bp}})$ are $\underline{\underline{\underline{ACL}}}$ and $\underline{\underline{\underline{\underline{\underline{ACLas}}}}}$ respectively
- the bottom-level functors $\underline{I}(\underline{\times})$, $\underline{I}(\underline{\ast})$, $\underline{I}(\underline{+})$, $\underline{I}(\underline{\perp})$ are locally continuous semi-functors on $\underline{\underline{\underline{\underline{ACLs}}}}$

The collecting interpretation is an approximating interpretation as follows from theorem 3.2:14 (and the facts listed before it). The use of $\underline{\underline{\underline{ACL}}}$ for $\underline{I}(\underline{\underline{B}})$ is in accord with the discussion in section 4.1 and the use of $\underline{\underline{\underline{\underline{\underline{ACLas}}}}}$ is because the tensor product is not a functor in general. The requirement that the bottom-level functors should be locally continuous semi-functors over $\underline{\underline{\underline{\underline{ACLs}}}}$ is motivated by:

Fact 4.2:1. If $F:\underline{\underline{\underline{\underline{ACLs}}}}^N \rightarrow \underline{\underline{\underline{\underline{ACLs}}}}$ is a locally continuous semi-functor and

(abs_i, con_i) are pairs of adjoined functions (relative to $\underline{\underline{ACLs}}$) then also $(F(abs_1,\ldots,abs_k), F(con_1,\ldots,con_k))$ is a pair of adjoined functions. ///

For an approximating interpretation $\underline{I}$ each $\underline{I}[\![gt]\!]$ is a locally continuous semi-functor over $\underline{\underline{ACLs}}$. This is by a structural induction where the case $gt=\Lambda_i$ is immediate. The remaining cases, except $gt=\underline{rec}X.gt'$, are straightforward because composition and tupling preserve the property of being a locally continuous semi-functor (as was already remarked in section 3.2). For the case $gt=\underline{rec}X.gt'$ it should be stressed that REC(G) is defined on morphisms by the formula

$$REC(G)(g_2,\ldots,g_N) = \bigsqcup_n r_n' \cdot (\lambda g.G(g,g_2,\ldots,g_N))^n(\bot) \cdot r_n^U$$

for embeddings r_n and r_n' as in section 2.2.

Lemma 4.2:2. If $G:\underline{\underline{ACLs}}^N \rightarrow \underline{\underline{ACLs}}$ is a locally continuous semi-functor then also $REC_i(G):\underline{\underline{ACLs}}^{N-1} \rightarrow \underline{\underline{ACLs}}$ is. ///

Proof We already know that REC(G) is a locally continuous functor over $\underline{\underline{ACLas}}$. Clearly it is locally continuous in general so it suffices to prove that

$$REC_1(G)(g_2 \cdot g_2', \ldots, g_N \cdot g_N') \sqsubseteq REC_1(G)(g_2,\ldots,g_N) \cdot REC_1(G)(g_2',\ldots,g_N')$$

holds in general and that equality holds when all g_i are additive. This is by a straightforward calculation. ///

Remark In lemma 2.2:10 a fixed point characterisation of REC(G)(...) was given. The proof given is not applicable here because it was assumed that G was a functor over the category where the g_i are morphisms. If $\underline{I}'[\![gt]\!]$ denotes the "functor" where REC(...) is defined according to the fixed point characterisation then this too

defines a locally continuous semi-functor over ACLs. It agrees with $\underline{I}[\![gt]\!]$ on abstraction functions and concretization functions. The result for abstraction functions is because they are lower adjoints and hence additive. Denote by abs^U the upper adjoint corresponding to abs. The result for concretization functions then is because they are upper adjoints and because the formula

$$\underline{I}[\![gt]\!](abs^U) = (\underline{I}[\![gt]\!]abs)^U = (\underline{I}'[\![gt]\!]abs)^U = \underline{I}'[\![gt]\!](abs^U)$$

...

is a consequence of the fact above. ///

RELATING THE INTERPRETATIONS

A central idea in abstract interpretation is to formalise the notion of when one data flow analysis is a safe approximation of another. Consider two approximating interpretations $\underline{I}$ and $\underline{J}$ and let $g \in \underline{I}[\![gt \underline{\rightarrow} gt']\!]$ and $h \in \underline{J}[\![gt \underline{\rightarrow} gt']\!]$ be functions to be related. For this purpose we assume that there is a family $(abs,con) = (abs_{gt},con_{gt})_{gt}$ of pairs of adjoined abstraction and concretization functions. As in the introduction the relation to be used is

$$sa_{gt \underline{\rightarrow} gt'}(g,h) \equiv g \cdot con_{gt} \sqsubseteq con_{gt'} \cdot h$$

This can be formulated in a number of equivalent ways, one being that $g(v) \sqsubseteq con_{gt'}(h(abs_{gt}(v)))$ holds for all elements v of $\underline{I}[\![gt]\!]$.

To extend this relation to all closed types t define the relation $\preceq_{con,t}$ on $\underline{I}[\![t]\!] \times \underline{J}[\![t]\!]$ by

$$g \preceq_{con,t} h \text{ iff } sim_t[\![sa]\!]()(g,h)$$

using the sim_t predicate defined in section 3.3. It follows from lemma 3.3:7 that this defines an admissible predicate. It further satisfies certain "reflexive" and "transitive" laws. To state these

let id denote the family $(id_{\underline{I}[\![gt]\!]})_{gt}$ of concretization functions and let $(con \cdot con')_{gt}$ be defined componentwise as $con_{gt} \cdot con'_{gt}$.

Lemma 4.2:3. If t is a closed and contravariantly pure type then

- $g \leq_{id,t} g$
- $g \leq_{con,t} h$ and $h \leq_{con',t} k$ implies $g \leq_{con \cdot con',t} k$
- $g \leq_{id,t} h$ implies $g \sqsubseteq h$ ///

Proof For the first and third result call an admissible predicate Q benevolent iff $Q(g,g)$ holds and $Q(g,h) \Rightarrow g \sqsubseteq h$ holds for all g and h. The result then follows from

if all Q_i are benevolent then so is $sim_t[\![sa]\!](Q_1,\ldots,Q_N)$

that is proved by structural induction upon types t such that $V \vdash t$ and CP(V,t) hold for some V with card(V) = N. The case $t=recX.t_0$ is by a straightforward numerical induction and in the case $t=t_1 \to t_2$ lemma 3.3:9 is used to show that $sim_{t_1}[\![sa]\!](Q_1,\ldots,Q_N)(v,w)$ holds iff v=w.

For the second result say that predicates Q and Q' force Q" iff for all g, h and k that Q(g,h) and Q'(h,k) implies Q"(g,k). (We do not merely define Q" as the relational composition of Q and Q' as this does not in general produce an admissible predicate.) The result then follows from

if for all i: Q_i and Q'_i force Q''_i

then $sim_t[\![\leq_{con}]\!](Q_1,\ldots,Q_N)$ and $sim_t[\![\leq_{con'}]\!](Q'_1,\ldots,Q'_N)$

force $sim_t[\![\leq_{con \cdot con'}]\!](Q''_1,\ldots,Q''_N)$

that is proved by structural induction upon types t such that $V \vdash t$ and CP(V,t) for some V with card(V)=N. The proof is quite straightforward. ///

Note The assumption about the type being contravariantly pure cannot simply be removed. To see this let $f \in \underline{C}[\![\underline{N \to N}]\!]$ be some finite element and define $h \in \underline{C}[\![(\underline{N \to N}) \to T]\!]$ by $h(f')=tt$ if f' is strictly greater than f and $h(f')=\perp$ otherwise. Then $h \not\preceq_{id,\underline{N\to N}\to T} h$ fails because $f \not\preceq_{id,\underline{N\to N}} f$ but $\perp \not\preceq_{id,T} tt$ is false. ///

We now define a relation $\preceq_{con}$ between approximating interpretations that is analogous to the condition expressed by lemma 3.3:11. Define $\underline{I} \preceq_{con} \underline{J}$ iff for all entries z of type t mentioned in the expression part it is the case that $\underline{I}(z) \preceq_{con,t} \underline{J}(z)$. The relation is intended to express that $\underline{J}$ is a safe approximation to $\underline{I}$. It satisfies the following "reflexive" and "transitive" laws:

Lemma 4.2:4. For approximating interpretations $\underline{I}$, $\underline{J}$ and $\underline{K}$:

- $\underline{I} \preceq_{id} \underline{I}$
- $\underline{I} \preceq_{con} \underline{J}$ and $\underline{J} \preceq_{con'} \underline{K}$ implies $\underline{I} \preceq_{con\cdot con'} \underline{K}$ ///

Proof Because of the previous lemma it suffices to show the result for entries z corresponding to functionals. The first result is immediate because $\underline{I}(z)$ is monotonic (in fact continuous). For the second result let abs be the lower adjoint corresponding to con and suppose

$$g_i \cdot con \cdot con' \sqsubseteq con \cdot con'' \cdot k_i$$

Define $h_i = abs \cdot g_i \cdot con$ so that $g_i \cdot con \sqsubseteq con \cdot h_i$ and $h_i \cdot con' \sqsubseteq con'' \cdot k_i$. Then

$$\underline{I}(z)(g_1,\ldots,g_n) \cdot con \cdot con' \sqsubseteq con \cdot \underline{J}(z)(h_1,\ldots,h_n) \cdot con' \sqsubseteq con \cdot con'' \cdot \underline{K}(z)(k_1,\ldots,k_n). \quad ///$$

The "transitivity" result means that data flow analyses can be built

in stages by "stepwise coarsening".

The relation really desired between approximating interpretations $\underline{I}$ and $\underline{J}$ is that $\underline{I}[\![e]\!] \leq_{con,t} \underline{J}[\![e]\!]$ for all expressions e. This is almost achieved:

Theorem 4.2:5.

$\underline{I}[\![e]\!] \leq_{con,t} \underline{J}[\![e]\!]$ for all e, t such that $\emptyset \vdash e:t$

$\Downarrow$

$\underline{I} \leq_{con} \underline{J}$

$\Downarrow$

$\underline{I}[\![e]\!] \leq_{con,t} \underline{J}[\![e]\!]$ for all e, t such that $\emptyset \vdash e:t$ and t does not mention the top-level smash product ///

Proof The first implication is because the metalanguage contains the expressions $take_i$, ..., $\lambda x:ft_1 \times \ldots \times ft_k.tuple(x\downarrow 1,\ldots,x\downarrow k)$, ..., f_i. The second implication is by theorem 3.3:14. ///

Note After the proof of lemma 4.2:3 it was said that $h \leq_{id,t} h$ may fail if t is not contravariantly pure. This is not contradicted by the theorem above even though $\underline{I} \leq_{id} \underline{I}$ always holds by lemma 4.2:4. This is because the h that causes problems cannot be obtained as $h=\underline{C}[\![e]\!]$ for some expression e. ///

Remark In the statement of the theorem it was assumed that the type t of the expression e does not use the top-level smash product. To show that this is essential we shall assume that the constants of contravariantly pure type include the truth value tt, the undefined function $\perp:\underline{N \to N}$ and the identity function $id:\underline{N \to N}$. Then

$$\underline{C}[\![\perp]\!] \leq_{id,\underline{N \to N}} \underline{C}[\![id]\!]$$

and

$$\underline{C}[\![tt]\!] \leq_{id,T} \underline{C}[\![tt]\!]$$

both hold but

$$\underline{C}[\![\text{take}_2(*\bot, \text{tt}*)]\!] \not\leq_{id,T} \underline{C}[\![\text{take}_2(*id, \text{tt}*)]\!]$$

fails. To overcome this problem we could redefine

$$g \leq_{con, gt \underline{\rightarrow} gt'} h \quad \text{iff} \quad g \cdot con_{gt} \sqsubseteq con_{gt'} \cdot h \text{ and } g = \bot \Leftrightarrow h = \bot$$

but then lemma 4.2:4 fails: it is not the case that $\underline{C} \leq_{id} \underline{C}$ because $\underline{C}(\text{cond})(g_1, g_2, g_3)$ may be $\bot$ without any of the g_i being $\bot$ and clearly $\underline{C}(\text{cond})(T, T, T)$ is not $\bot$. The solution chosen was to abandon the use of smash product and this is in line with the remark in section 2.1.

Another solution might be to replace $t ::= gt \underline{\rightarrow} gt'$ by $t ::= (gt \underline{\rightarrow} gt')_\bot$ as then the smash product will pose no problems. This may be motivated with the following discussion about least elements. Usually in denotational semantics the least element in a cpo (e.g. $N_\bot$) is "unreal" in the sense that when produced by a computation it really means that the computation did not terminate. On the other hand the least element in a function space is a "real" value namely the function that never terminates when executed. If this function is produced by a computation this is not the same as saying that the computation did not terminate. Therefore an "unreal" least element should be added below the least "real" element. ///

INDUCING AN INTERPRETATION

We now turn to ways of describing a data flow analysis. The obvious possibility is to specify all the components of an approximating interpretation. It is possible, however, to specify the expression part in a very "automatic" way by generalising the idea in /CoCo79/ of inducing predicate transformers. The type part of

the abstracting interpretation must still be specified in full.

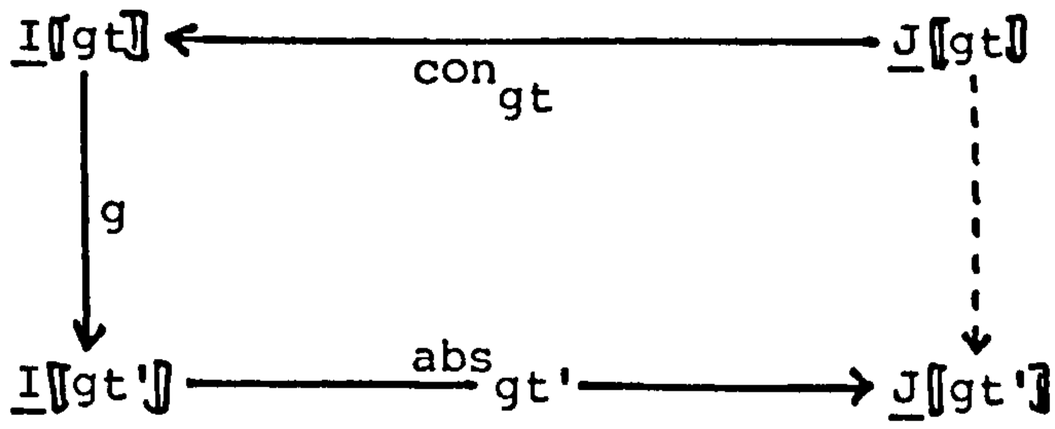
So let $\underline{I}$ be an approximating interpretation and $\underline{J}$ the type part of an approximating interpretation. Further assume a family

$$(abs,con) = (abs_{gt}:\underline{I}[\![gt]\!] \to \underline{J}[\![gt]\!] \ , \ con_{gt}:\underline{J}[\![gt]\!] \to \underline{I}[\![gt]\!])_{gt}$$

of adjoined abstraction and concretization functions (adjoined relative to $\underline{\underline{ACLs}}$). Consider now a function g in $\underline{I}[\![gt \underline{\to} gt']\!]$ and how to transport it to $\underline{J}[\![gt \underline{\to} gt']\!]$. An obvious possibility, and the one used in /CoCo79/, is to use

$$ind_{gt \underline{\to} gt'}(g) = abs_{gt'} \cdot g \cdot con_{gt}$$

as is illustrated by the diagram



Clearly $ind_{gt \underline{\to} gt'}(g)$ is a safe approximation to g, i.e.

$$g \leq_{con, gt \underline{\to} gt'} ind_{gt \underline{\to} gt'}(g)$$

Also it is the best approximation to g in that

$$g \leq_{con, gt \underline{\to} gt'} h \text{ iff } ind_{gt \underline{\to} gt'}(g) \sqsubseteq h$$

which may be generalised to

$$g \leq_{con \cdot con', gt \underline{\to} gt'} h \text{ iff } ind_{gt'}(g) \leq_{con', gt \underline{\to} gt'} h$$

Another way of phrasing these conditions is that $ind_{gt \underline{\to} gt'}(g)$ is as faithful a representation of g that the spaces in $\underline{J}$ allow.

The above process can be extended to all contravariantly pure and closed types t using the transformation $view_t[...]$ defined in

section 3.3. So define

$$induce_{(abs,con),t}(g) = view_t[ind]()(g)$$

As before $induce_{(abs,con),t}(g)$ is the best safe approximation of g.

Lemma 4.2:6. For any contravariantly pure and closed type t and family (abs,con) of pairs of adjoined functions

$$- g \leq_{con,t} induce_{(abs,con),t}(g)$$

$$- g \leq_{con\cdot con',t} h \text{ iff } induce_{(abs,con),t}(g) \leq_{con',t} h \qquad ///$$

Proof The first result is immediate by lemma 3.3:10 because the strictness of $con_{gt'}$ ensures that $abs_{gt'}\cdot g\cdot con_{gt}$ is strongly strict. For the second result "if" is immediate by the first result and lemma 4.2:3. "Only if" follows from the following result:

if k_i are strongly strict and $Q_i(g,h) \Rightarrow Q_i'(k_i(g),h)$

then, writing $g'=view_t[induce_{(abs,con)}](k_1,\ldots,k_N)(g)$,

it follows that

$$sim_t[\leq_{con\cdot con'},](Q_1,\ldots,Q_N)(g,h)$$

implies

$$sim_t[\leq_{con},](Q_1',\ldots,Q_N')(g',h)$$

The result is proved by induction on types t such that $V \vdash t$ and CP(V,t) for some set V with card(V)=N. We omit the details (but see the proof of lemma 3.3:10 for hints). ///

Because $induce_{(abs,con)}$ was defined using view an analogue of lemma 3.3:5 holds. From this it easily follows that

$$induce_{(id,id),t} = id_{\underline{I}[\![t]\!]}$$

$$induce_{(abs,con)\cdot(abs',con'),t} = induce_{(abs,con),t} \cdot induce_{(abs',con'),t}$$

where the composition of families of pairs of functions is defined by

$$(abs_{gt},con_{gt})_{gt} \cdot (abs'_{gt},con'_{gt})_{gt} = (abs_{gt} \cdot abs'_{gt}, con'_{gt} \cdot con_{gt})_{gt}$$

It remains to consider how to induce functionals as this is not found in /CoCo79/. So let z be a functional of type

$$t = (gt_1 \underline{\rightarrow} gt'_1) \times \ldots \times (gt_n \underline{\rightarrow} gt'_n) \rightarrow (gt \underline{\rightarrow} gt')$$

(which is closed but not contravariantly pure). We shall extend the definition of induce to include such types by defining

$$induce_{(abs,con),t}(\underline{I}(z)) =$$

$$\lambda(h_1,\ldots,h_n).\ abs_{gt'} \cdot \underline{I}(z)(con'_{gt_1} \cdot h_1 \cdot abs_{gt_1},\ldots) \cdot con_{gt}$$

This is best explained by considering the following diagrams. We begin with

$$\underline{J}[\![gt_i]\!] \xrightarrow{h_i} \underline{J}[\![gt'_i]\!]$$

which may be thought of as approximating

$$\underline{I}[\![gt_i]\!] \xrightarrow{g_i} con_{gt'_i} \cdot h_i \cdot abs_{gt_i} \rightarrow \underline{I}[\![gt'_i]\!]$$

The desired composition of these is

$$\underline{I}[\![gt]\!] \xrightarrow{\underline{I}(z)(g_1,\ldots,g_n)} \underline{I}[\![gt']\!]$$

which has

$$\underline{J}[\![gt]\!] \xrightarrow{abs_{gt'} \cdot \underline{I}(z)(g_1,\ldots,g_n) \cdot con_{gt}} \underline{J}[\![gt']\!]$$

as its best approximation. In other words, the extension of induce amounts to defining

$$induce_{(abs,con),t\rightarrow t'}(h) = induce_{(abs,con),t'} \cdot h \cdot induce_{(con,abs),t}$$

and this idea allows to extend the definition of induce to all closed types.

Putting it all together we define the approximating interpretation induce($\underline{I}$,(abs,con),$\underline{J}$) where $\underline{I}$ is an interpretation, $\underline{J}$ is the type part of an interpretation and (abs,con) is a gt-indexed family of adjoined abstraction and concretization functions. The type part is given by $\underline{J}$ and for an entry z (of type t) in the expression part:

$$\text{induce}(\underline{I},(\text{abs},\text{con}),\underline{J})(z) = \text{induce}_{(\text{abs},\text{con}),t}(\underline{I}(z))$$

That this gives the best approximation to $\underline{I}$ follows from:

Theorem 4.2:7.

- $\underline{I} \leq_{\text{con}} \text{induce}(\underline{I},(\text{abs},\text{con}),\underline{J})$
- $\underline{I} \leq_{\text{con}} J$ iff $\text{induce}(\underline{I},(\text{abs},\text{con}),\underline{J}) \leq_{\text{id}} \underline{J}$ ///

Proof Write $\underline{J}'=\text{induce}(\underline{I},(\text{abs},\text{con}),\underline{J})$. For the first result consider an entry z of type t. If t is of contravariantly pure type the result follows from lemma 4.2:6. Otherwise t is of the form $(gt_1 \underline{\to} gt_1') \times \ldots \times (gt_n \underline{\to} gt_n') \to (gt \underline{\to} gt')$. Suppose $g_i \leq_{\text{con},gt_i \underline{\to} gt_i'} h_i$ so that $g_i \sqsubseteq \text{con}_{gt_i'} \cdot h_i \cdot \text{abs}_{gt_i}$. Since $\underline{I}(z)$ is continuous we may calculate

$$\underline{I}(z)(g_1,\ldots,g_n)\cdot\text{con}_{gt} \sqsubseteq$$
$$\text{con}_{gt'}\cdot\text{abs}_{gt'}\cdot\underline{I}(z)(\text{con}_{gt_1'}\cdot h_1\cdot\text{abs}_{gt_1},\ldots)\cdot\text{con}_{gt} =$$
$$\text{con}_{gt'}\cdot\underline{J}'(z)(h_1,\ldots,h_n)$$

and this shows that $\underline{I}(z)(g_1,\ldots,g_n) \leq_{\text{con},gt \underline{\to} gt'} \underline{J}'(z)(h_1,\ldots,h_n)$.

The "if" part of the second result is by lemma 4.2:4 and the first result. For "only if" it follows by lemma 4.2:6 that it suffices to consider an entry z of type t as above. If $g_i \leq_{\text{id},gt_i \underline{\to} gt_i'} h_i$ then the result

$$\underline{J}'(z)(g_1,\ldots,g_n) \leq_{\text{id},gt \underline{\to} gt'} \underline{J}(z)(h_1,\ldots,h_n)$$

follows from the calculation

$$\underline{J}'(z)(g_1,\ldots,g_n) =$$

$$abs_{gt}\cdot\underline{I}(z)(con_{gt}\cdot g_1\cdot abs_{gt_1},\ldots)\cdot con_{gt} \sqsubseteq$$

(since $\underline{I} \leq_{con} \underline{J}$)

$$abs_{gt}\cdot con_{gt}\cdot\underline{J}(z)(h_1,\ldots,h_n) \sqsubseteq$$

$$\underline{J}(z)(h_1,\ldots,h_n) \qquad ///$$

It is straightforward to show the equalities

$$induce(\underline{I},(id,id),\underline{I}) = \underline{I}$$

$$induce(\ induce(\underline{I},(abs,con),\underline{J}),\ (abs',con'),\underline{K}) =$$

$$induce(\underline{I},(abs',con')\cdot(abs,con),\underline{K})$$

The last equality says that data flow analyses may be developed in a stepwise process and the resulting data flow analysis is as if it had been developed using only one step.

4.3 COMPOSITIONAL DEFINITION OF ABS AND CON

The definition of an interpretation specified that the meaning of bottom-level types was defined in a compositional way. It does not appear to be useful to have a notion of "long interpretation" where this is not so. Similarly it does not appear to be useful to consider abs_{gt} and con_{gt} that are not compositionally defined. The developments in the previous section were independent of how abs_{gt} and con_{gt} are specified so the question of how to specify them compositionally has been deferred to this section.

To be more specific let $\underline{I}$ and $\underline{J}$ be type parts of approximating interpretations. We shall assume as given pairs

$$(abs_{\underline{A}_i}:\underline{I}(\underline{A}_i)\to\underline{J}(\underline{A}_i)\ ,\ con_{\underline{A}_i}:\underline{J}(\underline{A}_i)\to\underline{I}(\underline{A}_i))$$

of adjoined (relative to $\underline{\underline{ACLs}}$) abstraction and concretization functions. The goal is to obtain a family

$$(abs_{gt}:\underline{I}[\![gt]\!]\rightarrow\underline{J}[\![gt]\!]\ ,\ con_{gt}:\underline{J}[\![gt]\!]\rightarrow\underline{I}[\![gt]\!])_{gt}$$

of adjoined functions (where gt ranges over the closed types only).

It may help to begin with sketching the simple case where $\underline{I}$ and $\underline{J}$ specify the same locally continuous semi-functor for the same bottom-level domain constructor. In the terminology of data flow analysis this implies that if $\underline{I}$ uses the relational method for $\underline{\times}$, i.e. $\underline{I}(\underline{\times})=\otimes$, then so does $\underline{J}$. Consider a closed type gt and let $A_{i_1}, \ldots, A_{i_n}$ be all the A_i's occurring in gt. Define the term $gt' = gt[X_j/\underline{A}_{i_j}]$ to be gt with $\underline{A}_{i_j}$ replaced by X_j. Then $\underline{I}[\![gt']\!]$ (as a semi-functor over $\underline{\underline{ACLs}}$) agrees with $\underline{J}[\![gt']\!]$ and we shall denote it by F_{gt}. Further $\underline{I}[\![gt]\!] = F_{gt}^{\cdot}(\underline{I}(\underline{A}_{i_1}),\ldots,\underline{I}(\underline{A}_{i_n}))$ and similarly for $\underline{J}[\![gt]\!]$. Defining

$$abs_{gt} = F_{gt}(abs_{\underline{A}_{i_1}},\ldots,abs_{\underline{A}_{i_n}})$$

$$con_{gt} = F_{gt}(con_{\underline{A}_{i_1}},\ldots,con_{\underline{A}_{i_n}})$$

then specifies functions of the right functionality and each (abs_{gt},con_{gt}) is an adjoined pair because F_{gt} is a locally continuous semi-functor. This corresponds roughly to what is done in /Nie81a/ for a somewhat simpler structure of types.

THE FRAMEWORK FOR CHANGE OF METHOD

In the general case the definition of abs_{gt} and con_{gt} is not so straightforward. As an example suppose that $\underline{I}$ uses the relational method for $\underline{\times}$, i.e. $\underline{I}(\underline{\times})=\emptyset$, whereas $\underline{J}$ uses the independent attribute method, i.e. $\underline{J}(\underline{\times})=\times$. A change in "method" takes place and some

additional information is needed to specify "how". The usual way of passing from one functor to another is by a natural transformation. This is complicated by $\underline{I}(\underline{x})$ not being a functor upon all of $\underline{\underline{\underline{\underline{ACLs}}}}$ and we shall therefore consider subcategories of $\underline{\underline{\underline{\underline{ACLs}}}}$. It is possible to give criteria for which subcategories that may be used but for the present purposes it suffices to consider the following two categories. Define $\underline{\underline{\underline{\underline{\underline{ACLsl}}}}}$ to be as $\underline{\underline{\underline{\underline{ACLs}}}}$ except that the morphisms included are only the lower adjoints of $\underline{\underline{\underline{\underline{ACLs}}}}$. Similarly $\underline{\underline{\underline{\underline{\underline{ACLsu}}}}}$ has upper adjoints as morphisms. Both are subcategories of $\underline{\underline{\underline{\underline{ACLs}}}}$ (but not sub cpo-categories). By fact 4.2:1 any locally continuous semi-functor specializes to a functor over both $\underline{\underline{\underline{\underline{\underline{ACLsl}}}}}$ and $\underline{\underline{\underline{\underline{\underline{ACLsu}}}}}$.

Let $\underline{\#}$ be any one of the bottom-level domain constructors $\underline{x}$, $\underline{*}$, $\underline{+}$, $\underline{\perp}$. To specify "how" the method is transformed from $\underline{I}(\underline{\#})$ to $\underline{J}(\underline{\#})$ we assume natural transformations

$$ABS_{\underline{\#}} \text{ from } \underline{I}(\underline{\#}) \text{ to } \underline{J}(\underline{\#}) \text{ over } \underline{\underline{\underline{\underline{\underline{ACLsl}}}}}$$

$$CON_{\underline{\#}} \text{ from } \underline{J}(\underline{\#}) \text{ to } \underline{I}(\underline{\#}) \text{ over } \underline{\underline{\underline{\underline{\underline{ACLsu}}}}}$$

satisfying that each

$$(ABS_{\underline{\#}}(L_1,\ldots,L_k) \ , \ CON_{\underline{\#}}(L_1,\ldots,L_k))$$

is a pair of adjoined functions. The definition of abs_{gt} and con_{gt} is very similar so a function $lengthen_{gt}$ is defined below. It will be used to define

$$abs_{gt} = lengthen_{gt}(\underline{I},(abs_{\underline{A}_i})_{\underline{A}_i},(ABS_{\underline{\#}})_{\underline{\#}},\underline{J})()$$

$$con_{gt} = lengthen_{gt}(\underline{J},(con_{\underline{A}_i})_{\underline{A}_i},(CON_{\underline{\#}})_{\underline{\#}},\underline{I})()$$

and it will be shown that (abs_{gt},con_{gt}) is a pair of adjoined functions of the right functionality.

The function lengthen_{gt} is defined by structural induction on types gt such that $V \vdash gt$ for some V with cardinality N. Let $\underline{I}$ and $\underline{J}$ be type parts of approximating interpretations and let

$$f_{\underline{A}_i} : \underline{I}(\underline{A}_i) \to \underline{J}(\underline{A}_i)$$
$$h_i : L_i \to L_i'$$

be morphisms of $\underline{\underline{\text{ACLs}}}$ and similarly

$$nt_{\underline{\#}}(L_1'', \ldots, L_k'') : \underline{I}(\underline{\#})(L_1'', \ldots, L_k'') \to \underline{J}(\underline{\#})(L_1'', \ldots, L_k'')$$

for all choices of objects L_i''. It is convenient to abbreviate

$$ln_{gt}(h_1, \ldots, h_N) = \text{lengthen}_{gt}(\underline{I}, (f_{\underline{A}_i})_{\underline{A}_i}, (nt_{\underline{\#}})_{\underline{\#}}, \underline{J})(h_1, \ldots, h_N)$$

The definition then is

$$ln_{\underline{A}_i}(h_1, \ldots, h_N) = f_{\underline{A}_i}$$
$$ln_X(h_1, \ldots, h_N) = h_i \quad \text{if X has index i in V}$$
$$ln_{gt_1 \underline{\#} \ldots \underline{\#} gt_k}(h_1, \ldots, h_N) = nt_{\underline{\#}}(\underline{J}[\![gt_1]\!](L_1', \ldots, L_k'), \ldots) \cdot \underline{I}(\underline{\#})(ln_{gt_1}(h_1, \ldots, h_N), \ldots)$$

for $\underline{\#}$ any one of $\underline{\times}$, $\underline{*}$, $\underline{+}$, $\underline{\perp}$

$$ln_{\underline{rec}X.gt_0}(h_1, \ldots, h_N) = \bigsqcup_n s_n \cdot LN_{gt_0,n} \cdot r_n^U$$

$\underline{\text{where}}$ s_n and r_n are the evident embeddings of $\underline{\underline{\text{ACLs}}}$ and

$LN_{gt_0,0} = \perp$, $LN_{gt_0,n+1} = ln_{gt_0}(h_1, \ldots, h_N, LN_{gt_0,n})$ assuming that X has index N+1 in $V \cup \{X\}$

Clearly this definition makes sense in $\underline{\underline{\text{ACLs}}}$ and defines a morphism of functionality

$$ln_{gt}(h_1, \ldots, h_N) : \underline{I}[\![gt]\!](L_1, \ldots, L_N) \to \underline{J}[\![gt]\!](L_1', \ldots, L_N')$$

That abs_{gt} is actually a lower adjoint and con_{gt} an upper adjoint is a consequence of the following:

$\underline{\text{Lemma}}$ 4.3:1. Let $\underline{\underline{\text{ACLsz}}}$ be one of $\underline{\underline{\text{ACLsl}}}$ and $\underline{\underline{\text{ACLsu}}}$. Let the $f_{\underline{A}_i}$ be

morphisms of $\underline{\underline{\underline{\underline{\underline{ACLsz}}}}}$ and $nt_{\underline{\underline{\#}}}$ a natural transformation over $\underline{\underline{\underline{\underline{\underline{ACLsz}}}}}$. Then $lengthen_{gt}(\underline{I},(f_{\underline{A}_i})_{\underline{A}_i},(nt_{\underline{\underline{\#}}})_{\underline{\underline{\#}}},\underline{J})()$ is a morphism of $\underline{\underline{\underline{\underline{\underline{ACLsz}}}}}$. ///

Proof The result follows from (using the abbreviation ln_{gt})

if $h_1,\ldots,h_N$ are morphisms of $\underline{\underline{\underline{\underline{\underline{ACLsz}}}}}$ then so is $ln_{gt}(h_1,\ldots,h_N)$

that is proved by structural induction on gt such that $V\vdash gt$ for some V with $card(V)=N$. The cases $gt=\underline{A}_i$ and $gt=X$ are straightforward. The case $gt=gt_1\underline{\#}\ldots\underline{\#}gt_k$ for $\underline{\#}$ one of $\underline{X}$, $\underline{*}$, $\underline{+}$ or $\underline{\perp}$ is because $\underline{I}(\underline{\#})$ is a functor over $\underline{\underline{\underline{\underline{\underline{ACLsz}}}}}$, $nt_{\underline{\underline{\#}}}$ is a natural transformation over $\underline{\underline{\underline{\underline{\underline{ACLsz}}}}}$ and because the composition of $\underline{\underline{\underline{\underline{\underline{ACLsz}}}}}$ morphisms gives such a morphism. Indeed if $\underline{\underline{\underline{\underline{\underline{ACLsz}}}}}$ is $\underline{\underline{\underline{\underline{\underline{ACLsl}}}}}$ and g^U denotes the upper adjoint of g then

$$ln_{gt}(h_1,\ldots,h_N)^U = \underline{I}(\underline{\#})(ln_{gt_1}(h_1,\ldots,h_N)^U,\ldots) \cdot nt_{\underline{\underline{\#}}}(\underline{J}(gt_1)(L_1',\ldots,L_N'),\ldots)^U$$

and similarly if $\underline{\underline{\underline{\underline{\underline{ACLsz}}}}}$ is $\underline{\underline{\underline{\underline{\underline{ACLsu}}}}}$. For the case $gt=\underline{rec}X.gt_0$ it is a straightforward induction to prove that $LN_{gt_0,n}$ is in $\underline{\underline{\underline{\underline{\underline{ACLsz}}}}}$ (because $\perp:U\to U$ is). To see that $\bigsqcup_n s_n\cdot LN_{gt_0,n}\cdot r_n^U$ is in $\underline{\underline{\underline{\underline{\underline{ACLsz}}}}}$ consider the case where $\underline{\underline{\underline{\underline{\underline{ACLsz}}}}}$ is $\underline{\underline{\underline{\underline{\underline{ACLsl}}}}}$ as the other case is similar. Let $LN_{gt_0,n}^U$ be the upper adjoint of $LN_{gt_0,n}$ and define $x = \bigsqcup_n r_n\cdot LN_{gt_0,n}^{U}\cdot s_n^U$. The calculations $ln_{gt}(h_1,\ldots,h_N)\cdot x\sqsubseteq id$ and $x\cdot ln_{gt}(h_1,\ldots,h_N)\sqsupseteq id$ show that $ln_{gt}(h_1,\ldots,h_N)$ is a lower adjoint of $\underline{\underline{\underline{\underline{ACLs}}}}$ with upper adjoint x.

///

Corollary 4.3:2. Given the conditions of the lemma, and assuming that $h_i:L_i\to L_i'$ are morphisms of $\underline{\underline{\underline{\underline{\underline{ACLsz}}}}}$ we have that

$$ln_{gt_1\underline{\#}\ldots\underline{\#}gt_k}(h_1,\ldots,h_N) = \underline{J}(\underline{\#})(ln_{gt_1}(h_1,\ldots,h_N),\ldots) \cdot nt_{\underline{\underline{\#}}}(\underline{I}(gt_1)(L_1,\ldots,L_N),\ldots)$$

///

Proof Use that $nt_{\underline{\underline{\#}}}$ is a natural transformation. ///

Given this corollary it is straightforward to adapt the proof of the lemma to show that (abs_{gt}, con_{gt}) is a pair of adjoined functions.

The lengthening process satisfies certain "reflexive" and "transitive" properties. To state these let $\underline{I}$, $\underline{J}$ and $\underline{K}$ be type parts of approximating interpretations and let $id_{\underline{\#}}$ be the natural transformation from $\underline{I}(\underline{\#})$ to $\underline{I}(\underline{\#})$ given by $id_{\underline{\#}}(L_1,\ldots) = id_{\underline{I}(\underline{\#})}(L_1,\ldots)$.

For a term gt such that $V \vdash gt$ and $card(V)=N$ let $i_1, \ldots, i_n$ be the increasing list of indices i such that $\underline{A}_i$ occurs in gt. Further let $gt[X_{N+j}/\underline{A}_{i_j}]$ denote the term where each $\underline{A}_{i_j}$ is replaced by X_{N+j} (and if necessary renaiming domain variables bound by a $\underline{rec}$).

$\underline{Lemma}$ 4.3:3. Let $\underline{\underline{ACLsz}}$ be one of $\underline{\underline{ACLsl}}$ or $\underline{\underline{ACLsu}}$. Let $nt=(nt_{\underline{\#}})_{\underline{\#}}$ and nt' be lists of natural transformations over $\underline{\underline{ACLsz}}$ and let $f=(f_{\underline{A}_i})_{\underline{A}_i}$, f' and f" be sequences of morphisms of $\underline{\underline{ACLsz}}$ and let h_i, h_i', h_i'' be morphisms of $\underline{\underline{ACLsz}}$. Then ("reflexivity")

$$lengthen_{gt}(\underline{I},f,id,\underline{I})(h_1,\ldots,h_N) =$$

$$\underline{I}[\![gt[X_{N+j}/\underline{A}_{i_j}]]\!](h_1,\ldots,h_N,f_{\underline{A}_1},\ldots,f_{\underline{A}_n})$$

and ("transitivity")

$$lengthen_{gt}(\underline{J},f'',nt',\underline{K})(h_1'',\ldots,h''_N)\cdot lengthen_{gt}(\underline{I},f',nt,\underline{J})(h_1',..)$$

$$= lengthen_{gt}(\underline{I},f''\cdot f',nt''\cdot nt',\underline{K})(h_1''\cdot h_1',\ldots,h_N''\cdot h_N') \qquad ///$$

The proof is by structural induction on gt and uses that $nt_{\underline{\#}}$ is a natural transformation from $\underline{I}(\underline{\#})$ to $\underline{J}(\underline{\#})$ and that $\underline{I}(\underline{\#})$ is a functor.

The definition of $lengthen_{\underline{rec}X.gt_0}$ was given in a form using embeddings. Using the previous lemma the following fixed point definition can be given much in the same way as was done for REC in section 2.2. This result will be used in the next section.

Lemma 4.3:4. Under the same conditions as in lemma 4.3:1 and its corollary we have

$$ln_{\underline{rec}X.gt_0}(h_1,\ldots,h_N) = LFP(\lambda h_{N+1}.\ i^{\cdot}ln_{gt_0}(h_1,\ldots,h_{N+1})^{\cdot}j^{-1})$$

Here it is assumed that X has index N+1 in $V_0\{X\}$, where $V\vdash\underline{rec}X.gt_0$, and that i and j are the evident isomorphisms from $gt[\underline{rec}X.gt/X]$ to $\underline{rec}X.gt$. ///

Proof The definition of the lefthand side is $\bigsqcup_n s_n^{\cdot}LN_{gt_0,n}^{\cdot}r_n^U$ and the righthand side is of the form $\bigsqcup_n G^n(\bot)$. To show the result it follows from $\bigsqcup_n r_n^{\cdot}r_n^U = id$ and $\bigsqcup_n s_n^{\cdot}s_n^U = id$ that it suffices to show either

$$s_n^{\cdot}LN_{gt_0,n} = G^n(\bot)^{\cdot}r_n \quad \text{or} \quad LN_{gt_0,n}^{\cdot}r_n^U = s_n^U{\cdot}G^n(\bot)$$

If the category in question (ACLsz) is ACLsl then the first result will be shown and when the category is ACLsu the second result will be shown.

The proofs are similar so suppose the category is ACLsl. The proof is by induction on n and the base case is trivial so consider n+1. We may calculate

$$G^{n+1}(\bot)^{\cdot}r_{n+1} =$$

(by section 2.2 $r_{n+1} = j^{\cdot}\underline{I}[\![gt_0]\!](id,\ldots,r_n)$)

$$i^{\cdot}ln_{gt_0}(h_1,\ldots,h_N,G^n(\bot))^{\cdot}\underline{I}[\![gt_0]\!](id,\ldots,r_n) =$$

(by the previous lemma because $G^n(\bot)$, h_i and r_n are lower adjoints)

$$i^{\cdot}ln_{gt_0}(h_1,\ldots,h_N,G^n(\bot))^{\cdot}lengthen_{gt_0}(\ldots)(id,\ldots,r_n) =$$

(by the previous lemma)

$$i^{\cdot}ln_{gt_0}(h_1,\ldots,h_N,G^n(\bot)^{\cdot}r_n) =$$

(by the inductive hypothesis and calculation as above)

$$s_{n+1}^{\cdot}LN_{gt_0,n+1}$$ ///

EXAMPLE CHANGES OF METHOD

The compositional definition of abs_{gt} and con_{gt} depends on the natural transformations $ABS_{\underline{\sharp}}$ and $CON_{\underline{\sharp}}$. When $\underline{I}(\underline{\sharp})$ and $\underline{J}(\underline{\sharp})$ are the same locally continuous semi-functor it is natural to use the identity natural transformation $id_{\underline{\sharp}}$. When they differ we shall consider three examples below. They do all have the flavour of changing from a "relational method" to an "independent attribute method". The latter methods may be desirable from a practical point of view because they are likely to result in faster although less precise data flow analyses (see /Jon81a/ for a formal result). Further it is mostly such methods that have previously been considered (in e.g. /Don78/). There is no clear definition of when a method is relational but we shall take it to mean "is as in the collecting semantics".

Example: The domain constructor $\underline{\times}$.

Here we investigate changing from the relational method $\underline{I}(\underline{\times}) = \otimes$ to the independent attribute method $\underline{J}(\underline{\times}) = \times$. The proposed transformations are (recalling section 3.2):

$$CON_{\underline{\times}}(L_1,\ldots,L_k) = cross_{\times}$$
$$= \lambda(l_1,\ldots,l_k).\ \{t \mid \exists b_1 \in B_{L_1},\ldots:$$
$$(\forall i: b_i \sqsubseteq l_i) \wedge t \sqsubseteq \otimes(b_1,\ldots,b_k)\}$$

$$ABS_{\times}(L_1,\ldots,L_k) = id^{\times}$$
$$= \lambda J.\ \bigsqcup\{(b_1,\ldots,b_k) \mid \otimes(b_1,\ldots,b_k) \in J\}$$

Clearly this defines morphisms in $\underline{\underline{ACLs}}$.

To see that the pairs of morphisms constitute pairs of (exactly) adjoined functions it suffices to perform the following two

calculations. First

$$ABS_{\underline{X}}(L_1,\ldots,L_k)\cdot CON_{\underline{X}}(L_1,\ldots,L_k) = id^{X}\cdot cross = id \subseteq id$$

where $id^{X}\cdot cross=id$ is because $L_1\otimes\ldots\otimes L_k$ is a tensor product with inclusion cross and extension of functions given by $\ldots^{X}$. Secondly

$$CON_{\underline{X}}(L_1,\ldots,L_k)\cdot ABS_{\underline{X}}(L_1,\ldots,L_k) =$$

$$cross\cdot id^{X}=$$

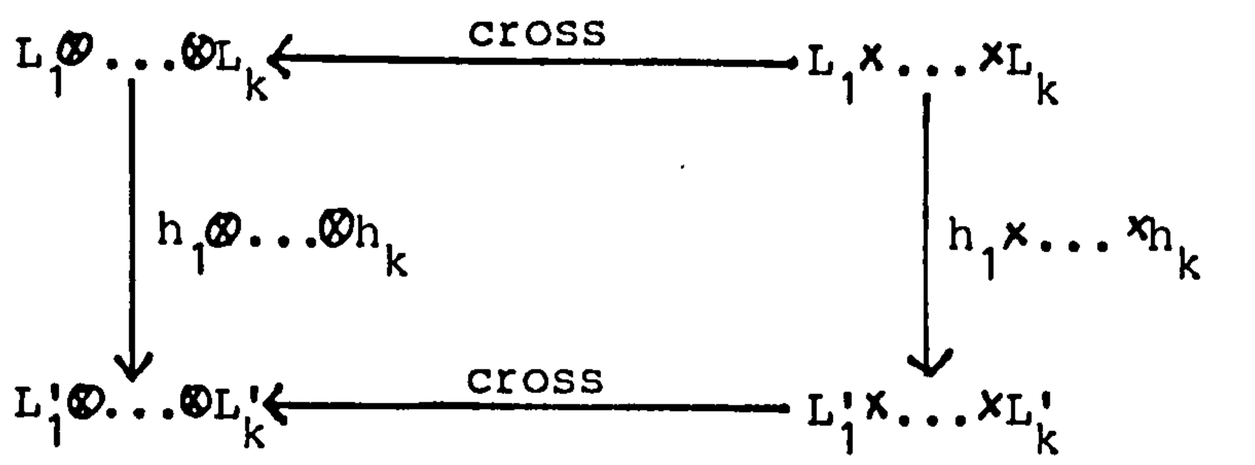
$$\lambda J.\{t \mid \exists b_1\in B_{L_1},\ldots: t\sqsubseteq\otimes(b_1,\ldots,b_k)$$

$$\wedge\ (b_1,\ldots,b_k)\sqsubseteq \bigsqcup\{(b_1',\ldots,b_k') \mid \otimes(b_1',\ldots,b_k')\in J\}\} \sqsupseteq$$

$$\lambda J.J = id$$

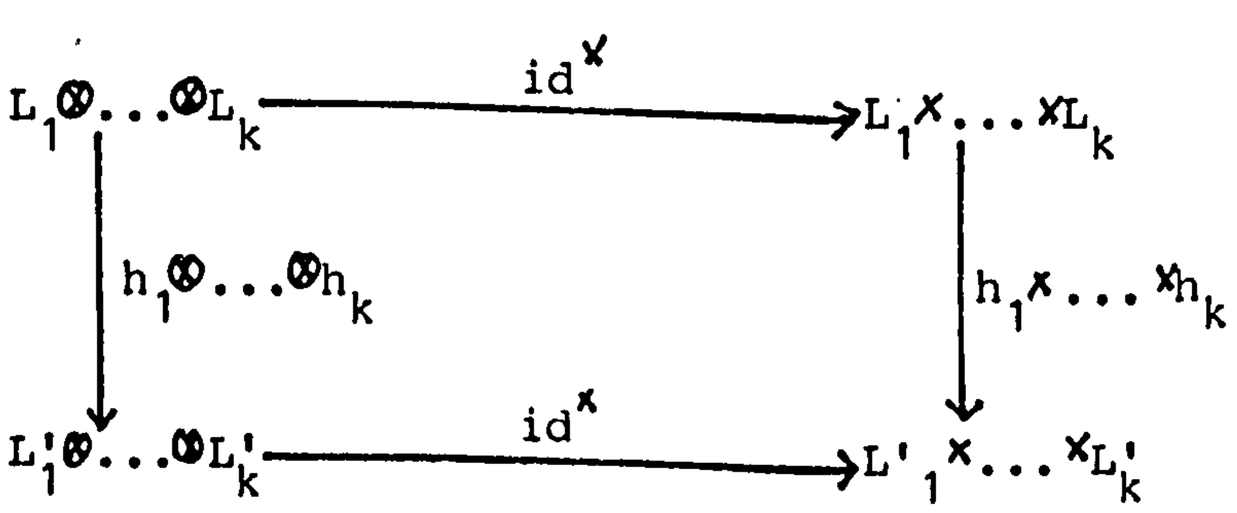
showing that the desired result holds.

That $CON_{\underline{X}}$ is a natural transformation over $\underline{\underline{ACLsu}}$ follows because the diagram



commutes for all $\underline{\underline{ACL}}$ morphisms h_i. To see this recall that $h_1\otimes\ldots\otimes h_k$ was defined as $(cross\cdot h_1\times\ldots\times h_k)^{X}$ and it was stated in section 3.2 that $f^{X}\cdot cross=f$ for all $\underline{\underline{ACL}}$ morphisms f.

To see that $ABS_{\underline{X}}$ is a natural transformation over $\underline{\underline{ACLsl}}$ it suffices to show that



commutes for all $\underline{\underline{\text{ACLa}}}$ morphisms h_i. Commutativity of the diagram is verified by the calculation

$$id^{\times\cdot}h_1\otimes\ldots\otimes h_k =$$

$$id^{\times\cdot}(cross\cdot h_1\times\ldots\times h_k) =$$

(because $id^{\times}$ is completely additive and it is straightforward to verify that $g\cdot f^{\times}=(g\cdot f)^{\times}$ whenever g is completely additive)

$$(id^{\times\cdot}cross\cdot h_1\times\ldots\times h_k)^{\times} =$$

(because $id^{\times\cdot}cross=id$)

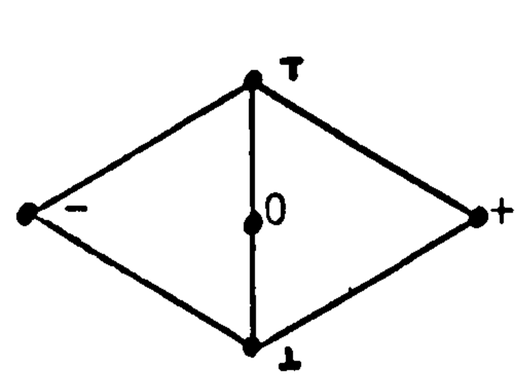
$$(h_1\times\ldots\times h_k\cdot id)^{\times} =$$

(as before because h_i is completely additive and therefore so is $h_1\times\ldots\times h_k$)

$$h_1\times\ldots\times h_k\cdot id^{\times}$$

Note The additivity of the h_i cannot simply be dispensed with. As an example take $L_1 = L_2 = L_1' = L_2' = V$ where

V =



was defined in section 3.1. Next define $h_1 = h_2 = \lambda x.x^2$ where $-^2=+$ etc. For $J = \{t \mid t\supseteq\otimes(-,-)\sqcup\otimes(+,+)\}$ we have $((h_1\times h_2)\cdot id^{\times})(J)=(T,T)$ whereas $(id^{\times\cdot}(h_1\otimes h_2))(J)=(+,+)$. ///

One could also consider investigating a change from the independent attribute method $\times$ back to the relational method $\otimes$. It is hard to do so, however, because it is hard to find a pair

$$(abs:L_1\times\ldots\times L_k\to L_1\otimes\ldots\otimes L_k\ ,\ con:L_1\otimes\ldots\otimes L_k\to L_1\times\ldots\times L_k)$$

of adjoined functions (relative to $\underline{\underline{\text{ACLs}}}$). This does not give cause for concern, however, as the proposed change does not seem to be of

interest for data flow analysis purposes.

Example: The domain constructor $\underline{\ast}$.

We first investigate changing from $\underline{I}(\underline{\ast})=\otimes$ to $\underline{J}(\underline{\ast})=\ast$. The proposed transformations are:

$$\begin{aligned}
CON_{\underline{\ast}}(L_1,\ldots,L_k) &= cross_{\ast} \\
&= \lambda(l_1,\ldots,l_k).\ \{t \mid \exists b_1 \in B_{L_1},\ldots: \\
&\qquad (\forall i: b_i \sqsubseteq l_i) \wedge\ t \sqsubseteq \otimes(b_1,\ldots,b_k)\} \\
ABS_{\underline{\ast}}(L_1,\ldots,L_k) &= id^{\ast} \\
&= \lambda J.\ \bigsqcup\{smash(b_1,\ldots,b_k) \mid \otimes(b_1,\ldots,b_k) \in J\}
\end{aligned}$$

Clearly this defines morphisms in $\underline{\underline{ACLs}}$.

By calculations similar to those given for $\underline{\times}$ it can be shown that the pairs $(ABS_{\underline{\ast}}(L_1,\ldots,L_k)\ ,\ CON_{\underline{\ast}}(L_1,\ldots,L_k))$ are (exactly) adjoined. Further, $CON_{\underline{\ast}}$ is a natural transformation from $\ast$ to $\otimes$ over $\underline{\underline{ACLsu}}$ because (similarly to $\underline{\times}$)

$$cross \cdot h_1 \ast \ldots \ast h_k = h_1 \otimes \ldots \otimes h_k \cdot cross$$

holds for all $\underline{\underline{ACLs}}$ morphisms h_i. Finally, $ABS_{\underline{\ast}}$ is a natural transformation from $\otimes$ to $\ast$ over $\underline{\underline{ACLsl}}$ because (similarly to $\underline{\times}$)

$$h_1 \ast \ldots \ast h_k \cdot id^{\ast} = id^{\ast} \cdot h_1 \otimes \ldots \otimes h_k$$

holds for all strongly strict $\underline{\underline{ACLas}}$ morphisms h_i (as then $h_1 \ast \ldots \ast h_k$ is completely additive).

Next we investigate changing from $\underline{I}(\underline{\ast})=\otimes$ to $\underline{J}(\underline{\ast})=\times$. For this define $ABS_{\underline{\ast}}$ and $CON_{\underline{\ast}}$ by the formulae explicitly exhibited above. It may be verified that $(ABS_{\underline{\ast}}(L_1,\ldots,L_k), CON_{\underline{\ast}}(L_1,\ldots,L_k))$ is a pair of adjoined functions (with $ABS_{\underline{\ast}}(L_1,\ldots,L_k) \cdot CON_{\underline{\ast}}(L_1,\ldots,L_k) = smash$). To show that $CON_{\underline{\ast}}$ is a natural transformation over $\underline{\underline{ACLsu}}$ it suffices to

show that

$$CON_{\underline{*}}(L_1',\ldots,L_k')\cdot h_1 \times \ldots \times h_k = h_1 \otimes \ldots \otimes h_k \cdot CON_{\underline{*}}(L_1,\ldots,L_k)$$

for all $\underline{\underline{ACLs}}$ morphisms h_i. To see this let $cross: L_1 * \ldots * L_k \to L_1 \otimes \ldots \otimes L_k$ be as above and note that

$$cross \cdot h_1 * \ldots * h_k \cdot smash = h_1 \otimes \ldots \otimes h_k \cdot cross \cdot smash$$

follows from the previous paragraph. The result then follows because $h_1 * \ldots * h_k \cdot smash$ equals $smash \cdot h_1 \times \ldots \times h_k$ and $cross \cdot smash$ equals $CON_{\underline{*}}(\ldots)$. Finally, to show that $ABS_{\underline{*}}$ is a natural transformation over $\underline{\underline{ACLsl}}$ it suffices to show that

$$h_1 \times \ldots \times h_k \cdot ABS_{\underline{*}}(\ldots) = ABS_{\underline{*}}(\ldots) \cdot h_1 \otimes \ldots \otimes h_k$$

holds for all strongly strict $\underline{\underline{ACLas}}$ morphisms h_i. This follows from the previous paragraph in much the same way as for $CON_{\underline{*}}$.

Example: The domain constructor $\underline{+}$.

We first define a locally continuous semi-functor $\oplus$. On objects

$$L_1 \oplus \ldots \oplus L_k = \{\bot,\top\} \cup \bigcup_i \{(i,d) \mid d \in L_i\}$$

which is partially ordered by

$$x \sqsubseteq y \text{ iff } x=\bot \text{ or } y=\top \text{ or for some } i, d, d' \text{ that } x=(i,d) \text{ and } y=(i,d') \text{ and } d \sqsubseteq d'$$

If all L_i are algebraic complete lattices then so is $L_1 \oplus \ldots \oplus L_k$ and $B_{L_1 \oplus \ldots \oplus L_k} = \{\bot\} \cup \bigcup_i B_{L_i}$. (If no new greatest element had been added and instead the $(i,\top)$ had been identified then algebraicity might be violated.) On morphisms

$$h_1 \oplus \ldots \oplus h_k = \lambda x. \begin{cases} \top & \text{if } x=\top \\ \bot & \text{if } x=\bot \\ (i,h_i(d)) & \text{if } x=(i,d) \end{cases}$$

This defines a strict and continuous function when all h_i are continuous. Further the definition depends continuously on the h_i. If all h_i are (completely) additive then so is $h_1 \oplus \ldots \oplus h_k$. (This would not have been the case if no new least element had been added and instead the $(i,\perp)$ had been identified.) It is then straightforward to verify that $\oplus$ is a locally continuous functor and semi-functor over $\underline{\underline{\underline{\underline{\text{ACLs}}}}}$.

Next we consider changing from the relational method $\underline{I}(\underline{+}) = \times$ to the independent attribute method $\underline{J}(\underline{+}) = \oplus$. The proposed transformations are

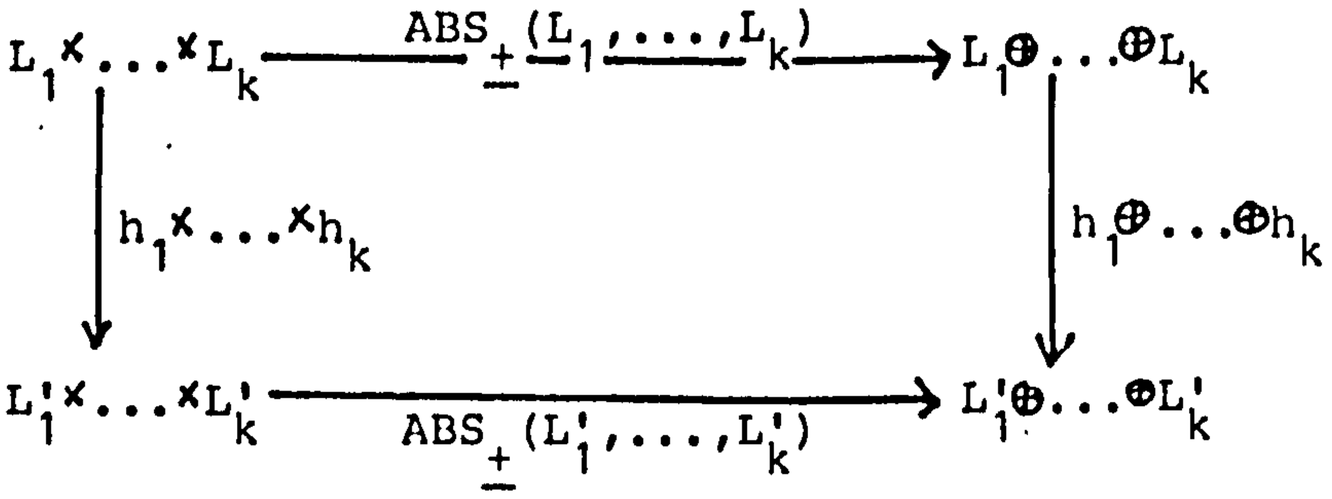
$$\text{CON}_{\underline{+}}(L_1,\ldots,L_k) = \lambda x. \begin{cases} (\perp,\ldots,\perp) & \text{if } x=\perp \\ (\top,\ldots,\top) & \text{if } x=\top \\ (\perp,\ldots,1,\ldots,\perp) & \text{if } x=(i,1) \end{cases}$$

$$\text{ABS}_{\underline{+}}(L_1,\ldots,L_k) = \lambda(1_1,\ldots,1_k). \begin{cases} \perp & \text{if } \forall i\colon 1_i=\perp \\ (i,1_i) & \text{if } \exists!i\colon 1_i=\perp \\ \top & \text{otherwise} \end{cases}$$

Clearly this specifies morphisms of $\underline{\underline{\underline{\underline{\text{ACLs}}}}}$ and it is straightforward to verify that the definitions specify pairs of adjoined functions. That $\text{CON}_{\underline{+}}$ is a natural transformation over $\underline{\underline{\underline{\underline{\underline{\text{ACLsu}}}}}}$ is because the diagram

$$\begin{array}{ccc} L_1 \times \ldots \times L_k & \xleftarrow{\text{CON}_{\underline{+}}(L_1,\ldots,L_k)} & L_1 \oplus \ldots \oplus L_k \\ \downarrow{\scriptstyle h_1 \times \ldots \times h_k} & & \downarrow{\scriptstyle h_1 \oplus \ldots \oplus h_k} \\ L'_1 \times \ldots \times L'_k & \xleftarrow{\text{CON}_{\underline{+}}(L'_1,\ldots,L'_k)} & L'_1 \oplus \ldots \oplus L'_k \end{array}$$

commutes for all $\underline{\underline{\underline{\underline{\text{ACLs}}}}}$ morphisms h_i such that $h_i(\top) = \top$. That $\text{ABS}_{\underline{+}}$ is a natural transformation over $\underline{\underline{\underline{\underline{\underline{\text{ACLsl}}}}}}$ is because the diagram



commutes for all strongly strict $\underline{\underline{ACLs}}$ morphisms h_i.

The only bottom-level domain constructor that has not been considered so far is $\underline{\perp}$. One idea might be to change from $\underline{I}(\underline{\perp}) = (\ )_{\perp}$ to $\underline{J}(\underline{\perp}) = Id$. It is unclear how this should be done because it is unclear how to find a pair

$$(abs: L_{\perp} \to L, con: L \to L_{\perp})$$

of adjoined functions (relative to $\underline{\underline{ACLs}}$). For this reason no "change of method" will be considered for lifting.

4.4 EXPECTED DEFINITIONS

The notion of inducing an interpretation allows for an automatic specification of the expression part of an approximating interpretation. It was shown in theorem 4.2:7 that this results in a very "precise" approximation of the original interpretation. There are circumstances, however, where it may be desirable to use more approximate versions of the functionals and some of the constants. These versions will be called expected definitions and in this section such definitions will be proposed and some of their properties studied.

Perhaps the most convincing example is composition $\square$ in the bottom-level metalanguage. Consider an approximating interpretation $\underline{I}$ with $\underline{I}(\square)(g_1,g_2) = g_1 \cdot g_2$ and let $\underline{J}$ be induced from $\underline{I}$. Then

$\underline{J}(\square)(h_1,h_2)$ is of the form

$$(abs \cdot con) \cdot h_1 \cdot (abs \cdot con) \cdot h_2 \cdot (abs \cdot con)$$

which may differ from $h_1 \cdot h_2$ when (abs,con) is not a pair of exactly adjoined functions. One may contemplate using an approximating interpretation $\underline{J}'$ that is mostly as $\underline{J}$ but has $\underline{J}'(\square)(h_1,h_2) = h_1 \cdot h_2$. Motivations for doing so may be conceptual, that "composition" should be composition, or based on implementation considerations such as whether it is easier to implement or faster to execute. Indeed previous formulations of abstract interpretation have always made this assumption. Having defined functional composition as the expected definition of $\square$ the question arises whether it is guaranteed to be safe to let both $\underline{I}$ and $\underline{J}'$ use the expected definitions. Formally the question is whether $\underline{I}(\square) = \lambda(g_1,g_2).\ g_1 \cdot g_2$ and $\underline{J}'(\square) = \lambda(h_1,h_2).\ h_1 \cdot h_2$ satisfy that

$$\underline{I}(\square) \leq_{con,ft^2 \to ft} \underline{J}'(\square)$$

By lemma 4.2:6 this is equivalent to asking whether $\underline{J}(\square) \leq_{id,ft^2 \to ft} \underline{J}'(\square)$. It is straightforward to verify that this is the case.

Additional examples

In the remainder of this section a similar treatment is given for all entries in the first and second components of the expresssion part of an interpretation. For some (e.g. in_i) the expected definition depends on which semi-functor is used to interpret some bottom-level domain constructor (e.g. $\underline{+}$). It will emerge that the collecting interpretation uses the expected definitions.

To facilitate the development the following assumptions are made. Let $\underline{I}$ and $\underline{J}$ be approximating interpretations such that $\underline{J}$ is induced from $\underline{I}$, i.e.

$$\underline{J} = \text{induce}(\underline{I},(\text{abs},\text{con}),\underline{J})$$

for a family $(\text{abs},\text{con}) = (\text{abs}_{gt},\text{con}_{gt})_{gt}$ of pairs of adjoined functions. For each of the bottom-level domain constructors (written $\underline{\#}$) we shall assume that the "change in method" from $\underline{I}(\underline{\#})$ to $\underline{J}(\underline{\#})$ is one of those considered in section 4.3. This means that the index for $(\underline{I}(\underline{\#}),\underline{J}(\underline{\#}))$ in the following table for $\underline{\#}$ must be a $\surd$.

$\underline{\perp}$	$\perp$	$\surd$

$\underline{+}$	$\times$	$\oplus$
$\times$	$\surd$	$\surd$
$\oplus$	%	$\surd$

$\underline{\times}$	$\otimes$	$\times$
$\otimes$	$\surd$	$\surd$
$\times$	%	$\surd$

$\underline{*}$	$\circledast$	$*$	$\times$
$\circledast$	$\surd$	$\surd$	$\surd$
$*$	%	$\surd$	%
$\times$	%	%	$\surd$

Further we shall assume that abs_{gt} and con_{gt} are specified compositionally as proposed in section 4.3. Taking abs_{gt} as an example this means that

$$\text{abs}_{gt} = \text{lengthen}_{gt}(\underline{I},(\text{abs}_{\underline{A}_i})_{\underline{A}_i},(\text{ABS}_{\underline{\#}})_{\underline{\#}},\underline{J})()$$

where $\text{ABS}_{\underline{\#}}$ was specified in section 4.3. (In particular $\text{ABS}_{\underline{\#}}$ is the identity transformation if $\underline{I}(\underline{\#})$ equals $\underline{J}(\underline{\#})$.) These assumptions suffice for most of the development but additional assumptions will be introduced to handle the expected definition of filter (used in cond) or when $\underline{J}$ uses tensor products ($\underline{J}(\underline{\times})=\otimes$ or $\underline{J}(\underline{*})=\circledast$).

The development is by means of a list of examples. The proposal of expected definitions may be viewed as defining

$$\underline{J}' = \text{expected-induce}(\underline{I},(\text{abs},\text{con}),\underline{J})$$

that is induce$(\underline{I},(\text{abs},\text{con}),\underline{J})$ modified so as to use the expected definitions. The proposals are guided by what holds in the collecting

interpretation. To show that it is safe to use the expected definitions amounts to proving

$$\underline{I} \leq_{con} \text{expected-induce}(\underline{I},(\text{abs},\text{con}),\underline{J})$$

and the relevant cases for that proof is given in each example.

Example: cond.

Consider the following expected definition

$$\underline{J}'(\text{cond})(h_1,h_2,h_3) = \text{strict}(h_2)^{\cdot}\underline{J}'(\text{filter}_{tt})(h_1) \sqcup \text{strict}(h_3)^{\cdot}\underline{J}'(\text{filter}_{ff})(h_2)$$

where $\underline{J}(\text{filter}_x)$ is not specified. There is no entry for filter in the expression part of an interpretation but this is not a serious problem because when $\underline{J}'(\text{cond})$ is of the above form $\underline{J}'(\text{filter}_{tt})(h_1)$ equals $\underline{J}'(\text{cond})(h_1,\text{id},\perp)$ and similarly for $\underline{J}'(\text{filter}_{ff})(h_1)$. If also $\underline{I}$ uses this expected definition it is straightforward to prove that

$$\underline{I}(\text{filter}_x) \leq_{con} \underline{J}'(\text{filter}_x) \quad \text{when x is tt and when x is ff}$$

$$\text{implies } \underline{I}(\text{cond}) \leq_{con} \underline{J}'(\text{cond})$$

Here types have been omitted from $\leq_{con}$ and this will often be the case in this section. ///

Example: fold and unfold.

It follows from section 2.2 that there is an isomorphism θ from $\underline{J}'[\![\text{gt}[\underline{\text{rec}}X.\text{gt}/X]]\!]$ to $\underline{J}'[\![\underline{\text{rec}}X.\text{gt}]\!]$. It is natural to expect the definitions

$$\underline{J}'(\text{fold}) = \theta \quad \text{and } \underline{J}'(\text{unfold}) = \theta^{-1}$$

If also $\underline{I}$ uses the expected definitions then

$$\text{con}_{\underline{\text{rec}}X.\text{gt}}{}^{\cdot}\underline{J}'(\text{fold}) = \underline{I}(\text{fold})^{\cdot}\text{ln}_{\text{gt}}(\text{con}_{\underline{\text{rec}}X.\text{gt}})$$

is an easy consequence of lemma 4.3:4. Further it can be shown by structural induction (see e.g. the proof of theorem 3.3:4) that

$$con_{gt[\underline{rec}X.gt/X]} = ln_{gt}(con_{\underline{rec}X.gt})$$

This shows that $\underline{I}(fold) \leq_{con} \underline{J}'(fold)$ and $\underline{I}(unfold) \leq_{con} \underline{J}'(unfold)$.

///

Example: lift and up.

The only functor considered is $\underline{J}'(\underline{\perp}) = (\ )_{\perp}$. This gives rise to the expected definitions

$$\underline{J}'(lift)(h) = \lambda v.\ def(v) \rightarrow h(down(v)), \perp$$

$$\underline{J}'(up) = up = \lambda v.(0,v)$$

If also $\underline{I}$ uses these expected definitions it is easy to show that

$$\underline{I}(up) \cdot con_{gt} = (con_{gt})_{\perp} \cdot \underline{J}'(up)$$

from which $\underline{I}(up) \leq_{con,gt \underline{\rightarrow} gt'} \underline{J}'(up)$ follows. It is equally straightforward to show that

$$\underline{I}(lift) \leq_{con} \underline{J}'(lift)$$

///

Example: case and in_i.

For $\underline{+}$ we have considered two semi-functors: $\times$ and $\oplus$. Naturally enough the expected definitions depend on the semi-functor. When $\underline{J}'(\underline{+}) = \times$ they are

$$\underline{J}'(case)(h_1,\ldots,h_k) = \lambda(l_1,\ldots,l_k).\ h_1(l_1) \sqcup \ldots \sqcup h_k(l_k)$$

$$\underline{J}'(in_i) = \lambda l_i.\ (\perp,\ldots,l_i,\ldots,\perp)$$

and when $\underline{J}'(\underline{+}) = \oplus$ they are given by

$$\underline{J}'(case)(h_1,\ldots,h_k)(\perp) = \bigsqcup_{i=1}^{k} h_i(\perp)$$

$$\underline{J}'(case)(h_1,\ldots,h_k)(j,l_j) = (\bigsqcup_{i=1}^{k} h_i(\perp)) \sqcup h_j(l_j)$$

$$\underline{J}'(case)(h_1,\ldots,h_k)(\tau) = \tau$$

$$\underline{J}'(in_i) = \lambda l.\ l=\perp \rightarrow \perp,\ (i,l)$$

The need for $\bigsqcup_{i=1}^{k} h_i(\bot)$ only arises because it has <u>not</u> been assumed that morphisms are strict (see the remark in section 2.3). The definition given for $\underline{J}'(in_i)$ could be replaced by the non-strict $\lambda 1.(i,1)$.

Suppose now that also $\underline{I}$ uses these expected definitions. The proof that $\underline{I}(in_i) \leq_{con} \underline{J}'(in_i)$ and $\underline{I}(case) \leq_{con} \underline{J}'(case)$ is by cases of whether $(\underline{I}(\underline{+}),\underline{J}'(\underline{+}))$ is $(\times,\times)$ or $(\times,\oplus)$ or $(\oplus,\oplus)$. For the first case $\underline{I}(in_i) \leq_{con} \underline{J}'(in_i)$ amounts to showing

$$\underline{I}(in_i)\cdot con_{gt_i} \sqsubseteq (con_{gt_1} \times \ldots \times con_{gt_k})\cdot \underline{J}'(in_i)$$

which is immediate. To show $\underline{I}(case) \leq_{con} \underline{J}'(case)$ assume that $g_i \cdot con_{gt_i} \sqsubseteq con_{gt} \cdot h_i$ and calculate

$$\underline{I}(case)(g_1,\ldots,g_k)\cdot con_{gt_1} \times \ldots \times con_{gt_k} =$$
$$\lambda(l_1,\ldots,l_k).\ g_1(con_{gt_1}(l_1)) \sqcup \ldots \sqcup g_k(con_{gt_k}(l_k)) \sqsubseteq$$
$$\lambda(l_1,\ldots,l_k).\ con_{gt}(h_1(l_1)) \sqcup \ldots \sqcup con_{gt}(h_k(l_k)) \sqsubseteq$$
$$con_{gt}\cdot \underline{J}'(case)(h_1,\ldots,h_k)$$

For the second case $\underline{I}(in_i) \leq_{con} \underline{J}'(in_i)$ is by a straightforward calculation. To show $\underline{I}(case) \leq_{con} \underline{J}'(case)$ assume that $g_i \cdot con_{gt_i} \sqsubseteq con_{gt} \cdot h_i$ and calculate (using the corollary to lemma 4.3:1)

$$\underline{I}(case)(g_1,\ldots,g_k)\cdot con_{gt_1} \times \ldots \times con_{gt_k} \cdot CON_{\underline{+}}(\ldots) \sqsubseteq$$
$$con_{gt} \cdot \lambda(l_1,\ldots,l_k).h_1(l_1) \sqcup \ldots \sqcup h_k(l_k) \cdot CON_{\underline{+}}(\ldots) \sqsubseteq$$
$$con_{gt}\cdot \underline{J}'(case)(h_1,\ldots,h_k)$$

For the third case $\underline{I}(in_i) \leq_{con} \underline{J}'(in_i)$ is again straightforward and for $\underline{I}(case) \leq_{con} \underline{J}'(case)$ assume that $g_i \cdot con_{gt_i} \sqsubseteq con_{gt} \cdot h_i$ and calculate

$$\underline{I}(case)(g_1,\ldots,g_k)\cdot con_{gt_1} \oplus \ldots \oplus con_{gt_k} =$$

$$\text{case}(g_1 \cdot \text{con}_{gt_1}, \ldots, g_k \cdot \text{con}_{gt_k}) \sqsubseteq$$
$$\text{case}(\text{con}_{gt} \cdot h_1, \ldots, \text{con}_{gt} \cdot h_k) \sqsubseteq$$
$$\text{con}_{gt} \cdot \underline{J}'(\text{case})(h_1, \ldots, h_k)$$

where case is defined by the same formula that defined $\underline{I}$(case) and $\underline{J}'$(case) (but the functionalities differ). ///

Example: tuple and take$_i$.

For $\underline{\times}$ the two semi-functors $\otimes$ and $\times$ have been considered. When $\underline{J}'(\underline{\times}) = \otimes$ the expected definitions will be

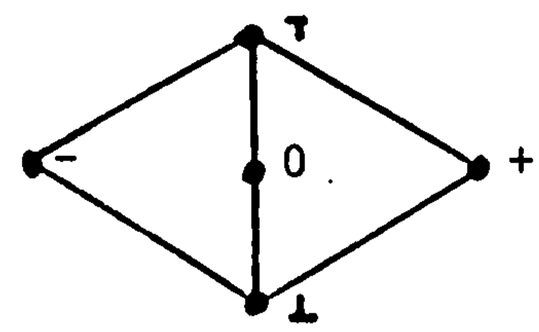
$$\underline{J}'(\text{tuple})(h_1, \ldots, h_k) = \text{lin}(\text{cross}_{\times} \cdot \lambda l.(h_1(l), \ldots, h_k(l)))$$
$$\underline{J}'(\text{take}_i) = \lambda l.\text{id}^{\times}(l)\downarrow i$$

as was used in the collecting interpretation. The choice for $\underline{J}'(\text{take}_i)$ is hardly unnatural and the definition of $\underline{J}'$(tuple) will be further motivated below. When $\underline{J}'(\underline{\times}) = \times$ it is natural to take

$$\underline{J}'(\text{tuple})(h_1, \ldots, h_k) = \lambda l.(h_1(l), \ldots, h_k(l))$$
$$\underline{J}'(\text{take}_i) = \lambda(l_1, \ldots, l_k).l_i$$

as the expected definitions.

To motivate the "expected definition" for $\underline{J}'$(tuple) when $\underline{J}'(\underline{\times}) = \otimes$ we shall consider the following example. Recall the cpo

V = 

where the intuitive meaning of the elements is given by the concretization function con:V→$\mathcal{P}(\{\ldots,-1,0,1,\ldots\}_{\perp})$ defined by con(⊥)={⊥}, con(−)={...,−1,⊥} etc. Next consider the "diagonal" function $\underline{J}'$(tuple)(id,id):V→V⊗V. When applied to ⊤ it will be

claimed that the desired result is $J_1=\{t \mid t \sqsubseteq \otimes(-,-) \sqcup \otimes(0,0) \sqcup \otimes(+,+)\}$ rather than e.g. $J_2=\{t \mid t \sqsubseteq \otimes(\tau,\tau)\}$. This is because J_1 expresses as precisely as possible that "the two components are really the same" quite unlike J_2. As an example of this J_2 contains $\otimes(-,+)$ but J_1 does not. If $\underline{J}'(\text{tuple})(h_1,\ldots,h_k)$ had simply been cross$\cdot\lambda 1.(h_1(1),\ldots,h_k(1))$ then J_2 would be obtained whereas the use of lin ensures that J_1 is obtained. The definition of lin given in section 3.3 was

$$\text{lin}(g) = \lambda 1.\bigsqcup\{g(i) \mid i \sqsubseteq 1 \wedge i \in IB\}$$

but for the purposes of section 3.3 PB could have been used instead of IB. This is not the case here because $IB_V = V-\{\tau\}$ whereas $PB_V=\{\perp\}$. Also the fact that IB is the set of finite and essential elements (see section 4.1) may be viewed as suggesting that IB is the right choice (or at least better than PB).

In the above discussion it was tacitly assumed that con is irreducibly generated. For had con been specified as above except that it was of functionality

$$\text{con}: V \to \mathcal{P}(\{\ldots,-1,0,1,\ldots,tt,ff\}_\perp)$$

then J_1 would intuitively be incorrect and J_2 should be used instead. It is therefore not surprising that the condition that abs specializes to abs:IB→IB (see lemma 4.1:4) will be used when showing $\underline{I}(\text{tuple}) \lesssim_{\text{con}} \underline{J}'(\text{tuple})$ in the case where $\underline{J}'(\underline{X})=\otimes$. To make it clear which parts of the developments can be performed without such assumptions this case will be postponed.

So suppose that $\underline{I}$ and $\underline{J}'$ both use the expected definitions and that $\underline{I}(\underline{X})=\otimes$ and $\underline{J}'(\underline{X})=X$. That $\underline{I}(\text{take}_i) \lesssim_{\text{con}} \underline{J}'(\text{take}_i)$ follows from calculating

$$\underline{I}(\text{take}_i)^{\cdot}\text{cross}^{\cdot}\text{con}_{gt_1} \times \ldots \times \text{con}_{gt_k} =$$

$$(\lambda 1.1{\downarrow}i)^{\cdot}\text{id}^{\times\cdot}\text{cross}^{\cdot}\text{con}_{gt_1} \times \ldots \times \text{con}_{gt_k} =$$

$$\text{con}_{gt_i}{}^{\cdot}(\lambda 1.1{\downarrow}i) = \text{con}_{gt_i}{}^{\cdot}\underline{J}'(\text{take}_i)$$

For $\underline{I}(\text{tuple}) \underline{\leq}_{\text{con}} \underline{J}'(\text{tuple})$ assume that $g_i{}^{\cdot}\text{con}_{gt} \sqsubseteq \text{con}_{gt_i}{}^{\cdot}h_i$ and calculate

$$\underline{I}(\text{tuple})(g_1,\ldots,g_k)^{\cdot}\text{con}_{gt} \sqsubseteq$$

$$\text{cross}^{\cdot}\lambda 1.((g_1{}^{\cdot}\text{con}_{gt})(1),\ldots,(g_k{}^{\cdot}\text{con}_{gt})(1)) \sqsubseteq$$

$$\text{cross}^{\cdot}\text{con}_{gt_1} \times \ldots \times \text{con}_{gt_k}{}^{\cdot}\lambda 1.(h_1(1),\ldots,h_k(1)) =$$

$$\text{con}_{gt_1 \underline{\times} \ldots \underline{\times} gt_k}{}^{\cdot}\underline{J}'(\text{tuple})(h_1,\ldots,h_k)$$

When both $\underline{I}(\underline{X})$ and $\underline{J}'(\underline{X})$ are $\times$ the calculations for $\underline{I}(\text{take}_i) \underline{\leq} \underline{J}'(\text{take}_i)$ and $\underline{I}(\text{tuple}) \underline{\leq} \underline{J}'(\text{tuple})$ are even more straightforward. ///

Remark Taking $\underline{\times}$ as an example one may consider whether

$$\text{con}_{gt_1 \underline{\times} \ldots \underline{\times} gt_k} = \underline{I}(\text{tuple})(\text{con}_{gt_1}{}^{\cdot}\underline{J}'(\text{take}_1),\ldots)$$

and similarly for $\text{abs}_{gt_1 \underline{\times} \ldots \underline{\times} gt_k}$. If true this would give a pleasant connection between the semi-functors and natural transformations and the functionals and constants. The equality holds in some cases but fails when $\underline{I}(\underline{\times}) = \otimes$ and $\underline{J}(\underline{\times}) = \times$: if $\text{con}_{gt_i} = \text{id}$ it reduces to

$$\text{cross} = \text{lin}(\text{cross})$$

which does not hold in general. ///

Example: smashtuple and smashtake_i.

For $\underline{*}$ the three semi-functors $\otimes$, $*$ and $\times$ have been considered. When $\underline{J}'(\underline{*}) = \circledast$ the expected definitions are

$$\underline{J}'(\text{smashtuple})(h_1,\ldots,h_k) =$$

$$\text{lin}(\text{cross}_{*}{}^{\cdot}\text{smash}^{\cdot}\lambda 1.(h_1(1),\ldots,h_k(1)))$$

$$\underline{J}'(\text{smashtake}_i) = \lambda 1.\text{id}^{*}(1){\downarrow}i$$

as in the collecting interpretation. When $\underline{J}'(\underline{*}) = \not*$ they are

$$\underline{J}'(\text{smashtuple})(h_1,\ldots,h_k) = \text{smash}\cdot\lambda 1.(h_1(1),\ldots,h_k(1))$$

$$\underline{J}'(\text{smashtake}_i) = \lambda 1.1\downarrow i$$

and when $\underline{J}'(\underline{*}) = \not\otimes$ they are

$$\underline{J}'(\text{smashtuple})(h_1,\ldots,h_k) = \lambda 1.(h_1(1),\ldots,h_k(1))$$

$$\underline{J}'(\text{smashtake}_i) = \lambda 1.1\downarrow i$$

If both $\underline{I}$ and $\underline{J}'$ use expected definitions then there are five cases of $(I(\underline{*}),\underline{J}'(\underline{*}))$ to consider in the proof of $\underline{I}(\text{smashtake}_i) \leq_{\text{con}} \underline{J}'(\text{smashtake}_i)$ and $\underline{I}(\text{smashtuple}) \leq_{\text{con}} \underline{J}'(\text{smashtuple})$. As in the previous example the case $(\circledast,\circledast)$ will be postponed. The remaining cases are tedious and are omitted as they are reasonably straightforward. ///

Making additional assumptions

It has already been said that the examples given so far constitute a definition of

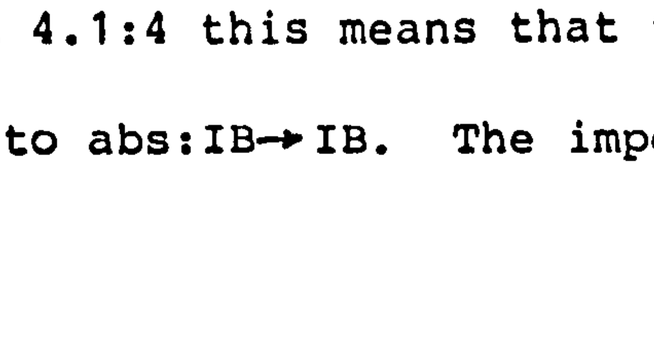
$$\underline{J}' = \text{expected-induce}(\underline{I},(\text{abs},\text{con}),\underline{J})$$

An exception is that no expected definitions for filter_x have been proposed. Furthermore, the examples constitute a proof of $\underline{I} \leq_{\text{con}} \underline{J}'$ except when $\underline{J}'(\underline{X}) = \otimes$ or $\underline{J}'(\underline{*}) = \circledast$. These omissions will now be rectified but additional assumptions about the abstraction functions abs_{gt} are needed for this.

In all these cases the same problem comes up. The desired result is of the form

$$\text{lin}(g) \leq_{\text{con,ft}} \text{lin}(h)$$

where $g \trianglelefteq_{con,ft} h$ is fairly straightforward to achieve. This is not, in general, sufficient for the desired result. As an example let



and $con: M \to \wp(N_\perp)$ be defined by $con(T) = N_\perp$, $con(1) = \{1, \perp\}$ etc. Further let $g(Y) = \{1 + (y \underline{div} 3) \mid y \in Y \wedge y \neq \perp\} \cup \{\perp\}$ and $h = abs \cdot g \cdot con$ for abs the lower adjoint of con. Then $g \trianglelefteq_{con,ft} h$ holds but $lin(g) \trianglelefteq_{con,ft} lin(h)$ fails because $(lin(g) \cdot con)(T) = N_\perp$ whereas $(con \cdot lin(h))(T) = \{1, \perp\}$. It is immediate that the concretization function con is not irreducibly generated. By lemma 4.1:4 this means that the abstraction function does not specialize to $abs: IB \to IB$. The importance of this is expressed in:

Lemma 4.4:1. Let $(abs_{gt}, con_{gt})_{gt}$ be a family of pairs of adjoined functions such that $g \trianglelefteq_{con, gt \underline{\to} gt'} h$. Then $lin(g) \trianglelefteq_{con, gt \underline{\to} gt'} lin(h)$ holds provided that abs_{gt} specializes to $abs_{gt}: IB \to IB$. ///

Proof Note that $g \trianglelefteq_{con, gt \underline{\to} gt'} h$ amounts to $abs_{gt'} \cdot g \sqsubseteq h \cdot abs_{gt}$. The result then follows from the calculation:

$$abs_{gt'} \cdot lin(g) =$$
$$\lambda l. \bigsqcup \{(abs_{gt'} \cdot g)(i) \mid i \sqsubseteq l \wedge i \in IB\} \sqsubseteq$$
$$\lambda l. \bigsqcup \{h(abs_{gt}(i)) \mid i \sqsubseteq l \wedge i \in IB\} \sqsubseteq$$
$$\lambda l. \bigsqcup \{h(abs_{gt}(i)) \mid abs_{gt}(i) \sqsubseteq abs_{gt}(l) \wedge i \in IB\} \sqsubseteq$$
$$lin(h) \cdot abs_{gt}$$

where it has been used that $abs_{gt'}$ is completely additive and that $i \in IB$ implies $abs_{gt}(i) \in IB$. ///

Example: tuple and $take_i$ (continued).

It remains to consider the case where $\underline{I}(\underline{x})$ and $\underline{J}'(\underline{x})$ are both $\otimes$.

For $\underline{I}(take_i) \leq_{con} \underline{J}'(take_i)$ it suffices to calculate:

$$\underline{I}(take_i)^{\cdot}con_{gt_1\underline{x}\ldots\underline{x}gt_k} =$$

$$(\lambda 1.1\downarrow i)^{\cdot}id^{x\cdot}(con_{gt_1}\otimes\ldots\otimes con_{gt_k}) =$$

(as in the example for $\underline{x}$ in section 4.3)

$$(\lambda 1.1\downarrow i)^{\cdot}(con_{gt_1}\times\ldots\times con_{gt_k}) =$$

$$(\lambda 1.1\downarrow i)^{\cdot}\lambda J.\sqcup\{(con_{gt_1}(b_1),\ldots)\mid \otimes(b_1,\ldots)\in J\} \sqsubseteq$$

$$(\lambda 1.1\downarrow i)^{\cdot}con_{gt_1}\times\ldots\times con_{gt_k}{}^{\cdot}\lambda J.\sqcup\{(b_1,\ldots)\mid \otimes(b_1,\ldots)\in J\} =$$

$$con_{gt_i}{}^{\cdot}\lambda 1.1\downarrow i^{\cdot}id^{x} =$$

$$con_{gt_i}{}^{\cdot}\underline{J}'(take_i)$$

For $I(tuple) \leq_{con} \underline{J}'(tuple)$ first assume $g_i{}^{\cdot}con_{gt} \sqsubseteq con_{gt_i}{}^{\cdot}h_i$ and calculate

$$cross^{\cdot}(\lambda 1.(g_1(1),\ldots,g_k(1)))^{\cdot}con_{gt} \sqsubseteq$$

$$cross^{\cdot}con_{gt_1}\times\ldots\times con_{gt_k}{}^{\cdot}\lambda m.(h_1(m),\ldots,h_k(m)) =$$

(by definition of $\otimes$ upon morphisms and by $f^{x\cdot}cross=f$)

$$con_{gt_1}\otimes\ldots\otimes con_{gt_k}{}^{\cdot}cross^{\cdot}\lambda m.(h_1(m),\ldots,h_k(m))$$

If it is assumed that all abs_{gt} specialize to $abs_{gt}:IB\rightarrow IB$ then the result follows by the previous lemma. ///

Example: smashtuple and $smashtake_i$ (continued).

The calculations and additional assumptions needed are much as above so the details will be omitted. ///

Example: filter.

In the expected definition for $\underline{J}'(cond)$ the functional $\underline{J}'(filter_x)$ was used and the proof of $\underline{I}(cond) \leq_{con} \underline{J}'(cond)$ was reduced to that of $\underline{I}(filter_x) \leq_{con} \underline{J}'(filter_x)$ for x=tt and x=ff. As the expected definition for $\underline{J}'(filter_x)$ one might take

$$\underline{J}'(\text{filter}_x)(h) = \text{lin}(\lambda 1.\ h(1) \sqsupseteq j_x \rightarrow 1\ ,\ \perp)$$

where $j_x \in \underline{J}'[\![\underline{T}]\!]$ is the representation in $\underline{J}'$ of the truthvalue x. (The collecting interpretation used $\{x\}_R$.) Rewriting this as

$$\underline{J}'(\text{filter}_x)(h) = \lambda 1.\ \bigsqcup\{i \in IB \mid i \sqsubseteq 1 \wedge j_x \sqsubseteq h(i)\}$$

shows that the effect of $\underline{J}'(\text{filter}_x)(h)(1)$ is to take the part of 1 upon which h "evaluates to" x.

Suppose further that $\underline{I}(\text{filter}_x)$ is also of this form but with $i_x \in \underline{I}[\![\underline{T}]\!]$ representing the truthvalue x. The proof of $\underline{I}(\text{filter}_x) \leq_{\text{con}} \underline{J}'(\text{filter}_x)$ will next be reduced to that of $\text{abs}_{\underline{T}}(i_x) \sqsupseteq j_x$. So assume that $g \cdot \text{con}_{gt} \sqsubseteq \text{con}_{\underline{T}} \cdot h$ and that $\text{abs}_{\underline{T}}(i_x) \sqsupseteq j_x$ and calculate:

$$(\lambda 1.\ g(1) \sqsupseteq i_x \rightarrow 1, \perp) \cdot \text{con}_{gt} =$$

$$\lambda m.\ (g \cdot \text{con}_{gt})(m) \sqsupseteq i_x \rightarrow \text{con}_{gt}(m), \perp \sqsubseteq$$

$$\text{con}_{gt} \cdot \lambda m.\ \text{con}_T(h(m)) \sqsupseteq i_x \rightarrow m, \perp =$$

$$\text{con}_{gt} \cdot \lambda m.\ h(m) \sqsupseteq \text{abs}_{\underline{T}}(i_x) \rightarrow m, \perp \sqsubseteq$$

$$\text{con}_{gt} \cdot \lambda m.\ h(m) \sqsupseteq j_x \rightarrow m, \perp$$

If it is further assumed that abs_{gt} specializes to $\text{abs}_{gt}: IB \rightarrow IB$ then the result follows by the previous lemma. ///

It remains to investigate when all abstraction functions abs_{gt} specialize to $\text{abs}_{gt}: IB \rightarrow IB$. The lemma below plays a key role in this. Recall that abs_{gt} was defined in section 4.3 as

$$\text{abs}_{gt} = \text{lengthen}_{gt}(\underline{I}, (\text{abs}_{\underline{A}_i})_{\underline{A}_i}, (\text{ABS}_{\underline{\#}})_{\underline{\#}}, \underline{J}')()$$

Let $\underline{\text{ACLsli}}$ be the subcategory of $\underline{\text{ACLsl}}$ of those morphisms abs that specialize to $\text{abs}: IB \rightarrow IB$.

Lemma 4.4:2. Let $\text{abs}_{\underline{A}_i}$ be morphisms of $\underline{\text{ACLsli}}$ and suppose that each

$ABS_{\underline{\#}}(L_1,\ldots,L_k)$ is in $\underline{\underline{ACLsli}}$. Further suppose that for each bottom-level domain constructor $\underline{\#}$ that $\underline{I}(\underline{\#})$ or $\underline{J'}(\underline{\#})$ specializes to a functor over $\underline{\underline{ACLsli}}$. Then all abs_{gt} are morphisms of $\underline{\underline{ACLsli}}$. ///

Proof The result follows from the stronger result:

if additionally $h_i:L_i \rightarrow L_i'$ are morphisms of $\underline{\underline{ACLsli}}$

then so is $lengthen_{gt}(\underline{I},abs,ABS,\underline{J'})(h_1,\ldots,h_N)$

that is proved by structural induction on gt such that $V \vdash gt$ for some V with card(V)=N. The cases $gt=\underline{A}_i$ and gt=X are immediate. The case $gt=gt_1\underline{\#}\ldots\underline{\#}gt_k$ (where $\underline{\#}$ is one of $\underline{\times}$, $\underline{*}$, $\underline{+}$ or $\underline{\perp}$) is straightforward if $\underline{I}(\underline{\#})$ is a functor over $\underline{\underline{ACLsli}}$ and by corollary 4.3:2 also if it is $\underline{J'}(\underline{\#})$ that is a functor over $\underline{\underline{ACLsli}}$.

For the case $gt=\underline{rec}X.gt_0$ we use the notation from section 4.3. It is a straightforward numerical induction to show that all $LN_{gt_0,n}$ are in $\underline{\underline{ACLsli}}$. It follows from lemma 4.3:1 that $h = \bigsqcup_n s_n \cdot LN_{gt_0,n} \cdot r_n^U$ is in $\underline{\underline{ACLsl}}$ so it suffices to show that h(i) is irreducible when $i \in IB$. First note that $h(i) = (s_n \cdot LN_{gt_0,n} \cdot r_n^U)(i)$ for sufficiently large n because h(i) is finite. Secondly we show that $r_n^U(i)$ is finite and irreducible for sufficiently large n. Note that $r_n(r_n^U(i)) = i$ when n is sufficiently large because i is finite and $\bigsqcup_n r_n \cdot r_n^U = id$. To see that $r_n^U(i)$ is finite suppose that $r_n^U(i) \sqsubseteq \bigsqcup_m d_m$. Then $i = r_n(r_n^U(i)) \sqsubseteq \bigsqcup_m r_n(d_m)$ so there exists an m such that $i \sqsubseteq r_n(d_m)$ and then $r_n^U(i) \sqsubseteq r_n^U(r_n(d_m)) = d_m$. To see that $r_n^U(i)$ is irreducible let $r_n^U(i) = d_1 \sqcup d_2$. Because r_n is additive $i = r_n(d_1) \sqcup r_n(d_2)$ so $i = r_n(d_k)$ for some k and it follows that $r_n^U(i) = d_k$.

Suppose now that n is large enough that $r_n^U(i)$ is finite and irreducible and h(i) equals $s_n(LN_{gt_0,n}(r_n^U(i)))$. Because $LN_{gt_0,n}$ is in $\underline{\underline{ACLsli}}$ it follows that $j = LN_{gt_0,n}(r_n^U(i))$ is finite and

irreducible. Now s_n is an embedding of $\underline{\underline{ACLsa}}$ and therefore $s_n(j)$ is irreducible. For if $s_n(j) = d_1 \cup d_2$ then $j = s_n^U(s_n(j)) = s_n^U(d_1) \cup s_n^U(d_2)$ because s_n^U is additive. So there exists a k such that $j = s_n^U(d_k)$ and it follows that $s_n(j) = s_n(s_n^U(d_k)) \sqsubseteq d_k$. But $s_n(j) \sqsupseteq d_k$ is trivial so $s_n(j) = d_k$ follows. Hence h(i) is irreducible. ///

The conditions of the above lemma may be made more concrete by using the assumptions about the connections between $\underline{I}(\underline{\#})$ and $\underline{J}'(\underline{\#})$ (which is $\underline{J}(\underline{\#})$) that were listed in the beginning of this section.

Lemma 4.4:3. The conditions of lemma 4.4:2 hold iff

- all $abs_{\underline{A}_i}$ specialize to $abs_{\underline{A}_i} : IB \to IB$, and
- $\underline{I}(\underline{x}) = \underline{J}'(\underline{x})$, and
- $\underline{I}(\underline{*}) = \underline{J}'(\underline{*}) \neq *$ ///

Proof The proof relies heavily upon information about the finite and irreducible elements so we begin with stating this information. It is straightforward to show that

$$IB_{L_1 \times \ldots \times L_k} = \bigcup_{i=1}^{k} \{\perp\} \times \ldots \times IB_{L_i} \times \ldots \times \{\perp\}$$
$$IB_{L_1 \oplus \ldots \oplus L_k} = \{\perp\} \cup \bigcup_{i=1}^{k} \{(i, b_i) \mid b_i \in IB_{L_i}\}$$
$$IB_{L_\perp} = \{\perp\} \cup \{up(b) \mid b \in IB_L\}$$

It is a consequence of lemma 3.2:19 that

$$IB_{L_1 \otimes \ldots \otimes L_k} = \{cross(b_1, \ldots, b_k) \mid \forall i: b_i \in IB_{L_i}\}$$
$$IB_{L_1 \circledast \ldots \circledast L_k} = \{cross(smash(b_1, \ldots, b_k)) \mid \forall i: b_i \in IB_{L_i}\}$$

We sketch the proof for $\otimes$. From theorem 3.1:3 it follows that $B_{L_1 \otimes \ldots \otimes L_k} = \{ \bigsqcup_{i=1}^{k} cross(b_1^i, \ldots, b_k^i) \mid b_j^i \in B_{L_j} \}$ and it is then easy to see that $\sqsubseteq$ holds above. To see that $\sqsupseteq$ holds note that if $b_i \in IB_{L_i}$ then $\otimes(b_1, \ldots, b_k)$ is irreducible by lemma 3.2:19 and if $cross(b_1, \ldots, b_k)$ was not irreducible this would give a contradiction.

Finally we state without proof that

$$IB_{L_1 * \ldots * L_k} = \{(\perp,\ldots,\perp)\} \cup \{(b_1,\ldots,b_k) \mid \exists i: \forall j: b_i \in IB_{L_i} \wedge b_i \neq \perp \wedge (\forall j: j \neq i \Rightarrow b_j \text{ is an atom})\}$$

where b_j is said to be an <u>atom</u> iff $b_j \sqsupseteq d \sqsupseteq \perp$ implies that $d=b_j$ or $d=\perp$.

Turning to the proof for "if" it is straightforward to show that all of $\otimes$, $\circledast$, $\times$, $\oplus$, $\perp$ give functors over $\underline{\underline{\text{ACLsli}}}$ (they give functors over $\underline{\underline{\text{ACLsl}}}$ because they are locally continuous semi-functors). For the natural transformations it is evident that $ABS_{\underline{\#}}(L_1,\ldots,L_k)$ is in $\underline{\underline{\text{ACLsli}}}$ when $\underline{I}(\underline{\#}) = \underline{J'}(\underline{\#})$ (because it is the identity). The only situation where this is not so is when $\underline{I}(\underline{+})=\times$ and $\underline{J'}(\underline{+})=\oplus$ and here it is straightforward to verify that $ABS_{\underline{+}}(L_1,\ldots,L_k)$ is in $\underline{\underline{\text{ACLsli}}}$.

Concerning the proof for "only if" it is straightforward to see that $\underline{I}(\underline{\times}) = \underline{J'}(\underline{\times})$ because otherwise

$$ABS_{\times}(L_1,\ldots,L_k) : L_1 \otimes \ldots \otimes L_k \rightarrow L_1 \times \ldots \times L_k$$

is not in $\underline{\underline{\text{ACLsli}}}$. Similarly $\underline{I}(\underline{*}) = \underline{J'}(\underline{*})$ is required. That $\underline{J'}(\underline{*}) \neq *$ is because $*$ does not give a functor over $\underline{\underline{\text{ACLsli}}}$. As an example define

$$L_1 = (\{0,1\},\geq)$$

$$L_2 = (\{0,1,2,\ldots,\infty\},\geq)$$

and $abs: L_1 \rightarrow L_2$ by $abs(0)=0$ and $abs(1)=\infty$. Then abs is in $\underline{\underline{\text{ACLsli}}}$ but $abs * abs: L_1 * L_1 \rightarrow L_2 * L_2$ is not. ///

In discussing the implications of these results let us assume that $\underline{I}$ is the collecting interpretation and let us forget filter for a moment. The motivation for assuming that abs_{gt} specializes to $abs_{gt}: IB \rightarrow IB$ was to handle the case where $\underline{J'}(\underline{x})=\otimes$ or $\underline{J'}(\underline{\times})=\circledast$. If

this property of abs_{gt} is to be guaranteed by lemma 4.4:2 it follows from lemma 4.4:3 that once a change from relational method to independent attribute method takes place for one of $\underline{X}$ or $\underline{*}$ then it should take place for the other as well. This is a bit restrictive but probably not a problem for practical applications.

Concerning filter the state of affairs is less ideal. Once a change from relational method to independent attribute method has taken place (for $\underline{X}$ or $\underline{*}$) then the expected definition no longer can be relied upon. It is, of course, possible to remove the use of lin but this means that a substantial amount of approximation takes place. It is not clear what other choice would be better.

Remark It might be wortwhile to investigate the following idea but to do so some changes in the metalanguage and the definition of interpretations will be needed. The idea is to work in the subcategory of $\underline{\underline{\underline{\underline{ACLs}}}}$ where the irreducible finite elements of an object are the least element and the atoms (they were defined in the previous proof). It is probably not overly restrictive to assume this for the bottom-level constant types $\underline{A}_i$. (In fact the "type structures" of /WaSh77/ are still included.) This does not work well with $\perp$, $\times$ and $\oplus$. So one might remove $\underline{\perp}$ and $\underline{X}$ from the metalanguage. (Luckily they were not used much in the examples in section 2.4.) One should disallow $\underline{J}'(\underline{*}) = X$ and redefine $\oplus$ to identify the $(i,\perp)$ with $\perp$. (It then does not give a semi-functor but it is still a functor that preserves lower adjoints.) This should be sufficient for a modified development where the second and third (!) conditions in lemma 4.4:3 vanish. Then one can still use lin in the expected definition of filter. ///

4.5 PRAGMATICS OF THE TENSOR PRODUCT

The distinction between relational data flow analysis methods and independent attribute methods is an intuitive one. It is relevant for all bottom-level domain constructors (except possibly $\underline{\perp}$) and we shall begin with $\underline{+}$. If L_i are "useful" approximations to $\wp(D_i)$ then which of $L_1 \times \ldots \times L_k$ or $L_1 \oplus \ldots \oplus L_k$ should be used to approximate $\wp(D_1 + \ldots + D_k)$? First, it is intuitively clear that use of $L_1 \times \ldots \times L_k$ is likely to give more precise results than use of $L_1 \oplus \ldots \oplus L_k$. (Note that if $\oplus$ had identified the $(i,\perp)$ with $\perp$ then there would be an abstraction function from $L_1 \times \ldots \times L_k$ to $L_1 \oplus \ldots \oplus L_k$ such that it and the corresponding concretization function is a pair of exactly adjoined functions.) Secondly, if $L_i = \wp(E_i)$ then $L_1 \times \ldots \times L_k$ is isomorphic to $\wp(E_1 + \ldots + E_k)$ and it is intuitively clear that this would give as precise results as possible (at least if the abstraction functions from $\wp(D_i)$ to $\wp(E_i)$ were defined using representation functions). For these reasons we shall say that use of $\times$ (for $\underline{+}$) gives a relational method and use of $\oplus$ (for $\underline{+}$) gives an independent attribute method. For $\underline{\times}$ it is $\otimes$ and $\times$ respectively and for $\underline{\ast}$ it is $\circledast$ and $\ast$.

Intuitively, there is one catch in the above. There might be many constructors that agree on powerdomains but disagree elsewhere. To be a bit more precise the above considerations only "determine" the relational method as a functor over PD that is defined as the image of $\underline{\underline{\text{ACC}}}$s under $\wp$. Given that the relational method for $\underline{+}$ is $\times$ over $\underline{\underline{\text{PD}}}$ it seems intuitively clear that it should be $\times$ also over $\underline{\underline{\text{ACL}}}$s. This is not necessarily as clear for the tensor products so in the remainder of this section we shall increase the confidence in the use of the tensor product $\otimes$. Such considerations have not

previously been made when formulating data flow analysis denotationally. For example /Don78/ only considers $\times$ for $\underline{\times}$ and + for $\underline{+}$.

The key idea is to define the notion of a "weak product". It is closely related to a tensor product with respect to "irreducibly generated" rather than "additive". It is convenient to perform the development in the following subcategory $\underline{\underline{\underline{\underline{ACLI}}}}$ of $\underline{\underline{\underline{ACL}}}$. An object L is included iff

$$B_L = \{ib_1 \sqcup \ldots \sqcup ib_n \mid n>0 \wedge \forall j: ib_j \in IB_L\}$$

and a morphism $f:L \to L'$ iff both L and L' are in $\underline{\underline{\underline{\underline{ACLI}}}}$. It is not overly restrictive to consider this category for it contains all powerdomains and also all objects L (of $\underline{\underline{\underline{ACL}}}$) such that there is a pair

$$(abs:\wp(D) \to L \ , \ con:L \to \wp(D))$$

of exactly adjoined functions with con irreducibly generated. Any additive morphism in this category is irreducibly generated. The weak product (in $\underline{\underline{\underline{\underline{ACLI}}}}$) of the objects L_i is an object $L_1 \boxtimes \ldots \boxtimes L_k$ and a morphism $unit:L_1 \times \ldots \times L_k \to L_1 \boxtimes \ldots \boxtimes L_k$ such that

- unit is seperately additive
- for each seperately irreducibly generated $f:L_1 \times \ldots \times L_k \to L$ there exists precisely one irreducibly generated $f^{\boxtimes}:L_1 \boxtimes \ldots \boxtimes L_k \to L$ such that $f^{\boxtimes} \cdot unit = f$.

Why is the property of a weak product more "intuitive" than that of the tensor product? That unit (or cross in the tensor product) is seperately additive can be paraphrased as follows. (Think of L_i as the complete lattice V with elements $\perp$, -, 0, +, $\top$.) It is possible to distinguish between combinations of arguments (e.g. $\{(+,-),(-,+)\}$ versus $\{(+,-),(+,+),(-,+)\}$) but this must not be used

to avoid the identifications in some argument that is prescribed by the least upper bound structure upon L_i (e.g. $\{(+,-),(-,-)\}$ cannot be distinguished from $\{(+,-),(-,-),(0,-)\}$). The difference between the weak product and the tensor product is the condition assumed of f and guaranteed for $f^{\boxtimes}$. The problem with the tensor product is that seperate additivity is unlikely to hold for f. (As an example consider $f:V\to V$ defined informally by $f(x)=x^2$. This function is not seperately additive so the tensor product property seems useless.) The condition that f is seperately irreducibly generated, i.e.

$$f(l_1,\ldots,l_k) = \bigsqcup\{f(ib_1,\ldots,ib_k) \mid \forall j:\ ib_j \sqsubseteq l_j \wedge ib_j \in IB_{L_j}\}$$

is likely to hold much more often. (But one should be careful for the property is not preserved by composition.) It says that f is given as the best (i.e. $\sqcup$) description of the results of <u>all</u> combinations of essential arguments (i.e. $(ib_1,\ldots,ib_k)$) that is permitted by the actual argument.

The purpose of $f^{\boxtimes}$ and $L_1\boxtimes\ldots\boxtimes L_k$ then is to allow more precision in <u>which</u> combinations to consider. This is made clearer by

<u>Lemma</u> 4.5:1. If $L_1\boxtimes\ldots\boxtimes L_k$ and unit is a weak product then

- $IB_{L_1\boxtimes\ldots\boxtimes L_k} = \{unit(ib_1,\ldots,ib_k) \mid \forall j:\ ib_j \in IB_{L_j}\}$
- $f^{\boxtimes}(x) = \bigsqcup\{f(ib_1,\ldots,ib_k) \mid unit(ib_1,\ldots,ib_k) \sqsubseteq x \wedge \forall j:\ ib_j \in IB_{L_j}\}$

///

<u>Proof</u> We begin with showing $\sqsubseteq$ in the first result. For the sake of contradiction suppose that ib is an element of the lefthand side but not of the righthand side. Define $L=\{\tau\}_\perp$ and $f:L_1\times\ldots\times L_k\to L$ by

$$f(l_1,\ldots,l_k)=\perp \text{ iff } \forall(ib_1,\ldots,ib_k)\in IB_{L_1}\times\ldots\times IB_{L_k}:$$
$$(ib_1,\ldots,ib_k)\sqsubseteq(l_1,\ldots,l_k) \Rightarrow unit(ib_1,\ldots,ib_k)\sqsubseteq ib$$

This gives a continuous and seperately irreducibly generated function

(but it is not seperately additive). Next define $g_i: L_1 \boxtimes \ldots \boxtimes L_k \to L$ by

$$g_1(x) = \bot \text{ iff } x \sqsubseteq ib$$

$$g_2(x) = \bot \text{ iff } x \sqsubseteq ib \wedge x \neq ib$$

Both g_1 and g_2 are continuous and additive functions. (For g_2 because ib is finite and irreducible.) So g_1 and g_2 are irreducibly generated because $L_1 \boxtimes \ldots \boxtimes L_k$ is an object of $\underline{\underline{ACLI}}$. To show that $g_i \cdot unit = f$ it suffices by continuity to show that $g_i(unit(b_1,\ldots,b_k)) = f(b_1,\ldots,b_k)$ holds when $b_j \in B_{L_j}$. For this note that

$$unit(b_1,\ldots,b_k) \sqsubseteq ib$$

holds iff

$$\forall (ib_1,\ldots) \in IB_{L_1} \times \ldots : (ib_1,\ldots) \sqsubseteq (b_1,\ldots) \Rightarrow unit(ib_1,\ldots) \sqsubseteq ib$$

because unit is seperately additive and L_i are objects of $\underline{\underline{ACLI}}$ (and in the case of g_2 because of the way ib was chosen). By the weak product property $g_1 = f^{\boxtimes} = g_2$ but this contradicts $g_1 \neq g_2$.

For $\supseteq$ let $ib_j \in IB_{L_j}$ and define $L = \{\tau\}_\bot$ and $f: L_1 \times \ldots \times L_k \to L$ by $f(l_1,\ldots,l_k) = \tau$ iff $\forall j: l_j \sqsupseteq ib_j$. This defines a continuous and seperately irreducibly generated function. Further

$$\tau = f(ib_1,\ldots,ib_k) =$$

(by $f^{\boxtimes}$ irreducibly generated and $f^{\boxtimes} \cdot unit = f$)

$$\bigsqcup \{ f^{\boxtimes}(ib) \mid ib \sqsubseteq unit(ib_1,\ldots,ib_k) \wedge ib \in IB \} =$$

(by $\subseteq$ above and $f^{\boxtimes} \cdot unit = f$)

$$\bigsqcup \{ f(ib'_1,\ldots,ib'_k) \mid unit(ib'_1,\ldots;ib'_k) \sqsubseteq unit(ib_1,\ldots,ib_k)$$
$$\wedge (\forall j: ib'_j \in IB_{L_j}) \wedge unit(ib'_1,\ldots,ib'_k) \in IB \}$$

Let $ib'_j \in IB_{L_j}$ be chosen such that $unit(ib'_1,\ldots,ib'_k) \in IB$ and $f(ib'_1,\ldots,ib'_k) = \tau$ and $unit(ib'_1,\ldots,ib'_k) \sqsubseteq unit(ib_1,\ldots,ib_k)$. Then

$(ib'_1,\ldots,ib'_k) \exists (ib_1,\ldots,ib_k)$ so by monotonicity of unit it follows that

$$unit(ib_1,\ldots,ib_k) = unit(ib'_1,\ldots,ib'_k) \in IB$$

The second result is an immediate consequence of the first. ///

The study of the tensor product has not been in vain, however.

Lemma 4.5:2. The tensor product is a weak product (with unit = cross and $f^{\boxtimes} = f^{\times}$). ///

Proof It follows from lemma 3.2:19 that $L_1 \otimes \ldots \otimes L_k$ is an object of $\underline{\underline{ACLI}}$ with $\dot{IB}_{L_1 \otimes \ldots \otimes L_k}$ being the set of elements $cross(ib_1,\ldots,ib_k)$ for $ib_j \in IB_{L_j}$. It is then straightforward that cross is a seperately additive morphism of $\underline{\underline{ACLI}}$. Next let $f:L_1 \times \ldots \times L_k \to L$ be seperately irreducibly generated. Uniqueness of $f^{\boxtimes}$ is by

$$f^{\boxtimes}(x) =$$

($f^{\boxtimes}$ irreducibly generated, $f^{\boxtimes} \cdot cross = f$ and information about IB)

$$\bigsqcup\{f(ib_1,\ldots,ib_k) \mid \forall j:\ ib_j \in IB_{L_j} \wedge cross(ib_1,\ldots,ib_k) \sqsubseteq x\} =$$

($\sqsubseteq$ obvious and $\sqsupseteq$ by f seperately irreducibly generated)

$$\bigsqcup\{f(b_1,\ldots,b_k) \mid \forall j:\ b_j \in B_{L_j} \wedge cross(b_1,\ldots,b_k) \sqsubseteq x\} =$$

$$f^{\times}(x)$$

For existence define $f^{\boxtimes}$ to be $f^{\times}$. Clearly $f^{\times}$ is a continuous function such that $f^{\times} \cdot cross = f$. It is immediate by the above calculation that $f^{\times}$ is irreducibly generated. ///

The tensor product is not the only weak product.

Example Recall that the elements of $V \otimes V$ are directed ideals J of terms t (e.g. $t = \otimes(+,+) \sqcup \otimes(-,-)$). Define the term $add_1(t)$ as

$$t \sqcup \bigsqcup_{b_i \neq \perp \neq b'_i} \otimes(b_i,b'_i) \text{ in } t\ \otimes(b_1 \sqcup b_2, b'_1 \sqcup b'_2)$$

(so that $add_1(\otimes(+,+)\cup\otimes(-,-)) = \otimes(+,+)\cup\otimes(-,-)\cup\otimes(\top,\top)$). Define the directed ideal $add_2(J)$ as

$$\{t \mid \exists t' \in J:\ t \sqsubseteq add_1(t')\}$$

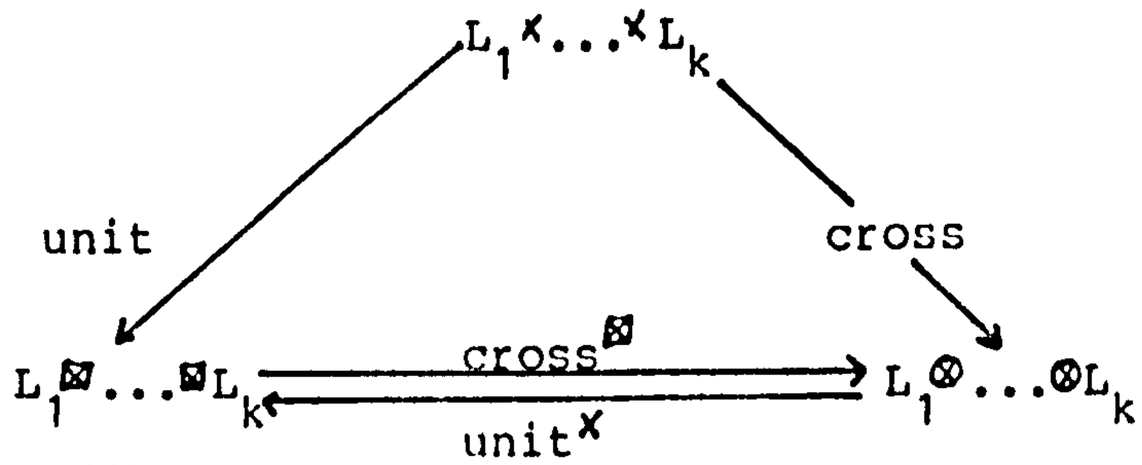
and note that add_2 is monotonic. By /Tar55/ the set of fixed points of $add_2: V\otimes V \to V\otimes V$ is a complete lattice (when ordered by $\sqsubseteq$ as in $V\otimes V$) which will be denoted $V\boxtimes V$. Then $add_3(J) = \bigcup_{n=0}^{\infty} add_2^{\,n}(J)$ is the least fixed point of add_2 that contains J. Defining unit = add_3 'cross the claim is that this gives a weak product with $f^{\boxtimes}$ being the restriction of $f^{\times}$ to $V\boxtimes V$. Clearly $V\boxtimes V$ is a proper subset of $V\otimes V$ and therefore is not (isomorphic to) the tensor product.

It is slightly complicated to verify the claim. Since the result is not of profound interest only a very brief sketch will be given. First $IB_{V\boxtimes V} \supseteq IB_{V\otimes V}$ is shown and this is used to show that $V\boxtimes V$ is an ACLI object. This again implies that $IB_{V\boxtimes V} = IB_{V\otimes V}$. Since add_3 is additive it is easy to see that unit satisfies the condition. It is straightforward to verify that the suggested $f^{\boxtimes}$ works and that no other possibilities exist. ///

The example above gives a weak product that is somehow simpler than the tensor product: it has fewer elements and so allows for less precision in which combinations of arguments that can be distinguished. In a certain sense this holds in general.

Lemma 4.5:3. For each weak product $L_1\boxtimes\ldots\boxtimes L_k$ there exist functions $abs: L_1\otimes\ldots\otimes L_k \to L_1\boxtimes\ldots\boxtimes L_k$ and $con: L_1\boxtimes\ldots\boxtimes L_k \to L_1\otimes\ldots\otimes L_k$ such that (abs,con) is a pair of exactly adjoined functions (relative to ACLs). ///

Proof Let unit and $f^{\boxtimes}$ refer to $L_1 \boxtimes \ldots \boxtimes L_k$ and cross and $f^{\times}$ to $L_1 \otimes \ldots \otimes L_k$. Recall that in ACLI every additive function is irreducibly generated. Consider the diagram



We shall verify the conditions for abs=$\text{cross}^{\boxtimes}$ and con=$\text{unit}^{\times}$.

It is immediate that id·unit equals $\text{unit}^{\times} \cdot \text{cross}^{\boxtimes} \cdot \text{unit}$. Further $\text{unit}^{\times}$ is completely additive: it is additive by the tensor product property and is strict because unit is. Clearly $\text{cross}^{\boxtimes}$ is irreducibly generated and it follows that so is $\text{unit}^{\times} \cdot \text{cross}^{\boxtimes}$. It follows that both id and $\text{unit}^{\times} \cdot \text{cross}^{\boxtimes}$ are candidates for $\text{unit}^{\boxtimes}$ and since $L_1 \boxtimes \ldots \boxtimes L_k$ is a weak product they are equal.

Next we show that $\text{id} \cdot \text{cross} = \text{lin}(\text{cross}^{\boxtimes} \cdot \text{unit}^{\times}) \cdot \text{cross}$. For this calculate

$$\text{id} \cdot \text{cross} =$$

$$\text{cross}^{\boxtimes} \cdot \text{unit}^{\times} \cdot \text{cross} \sqsupseteq$$

$$\text{lin}(\text{cross}^{\boxtimes} \cdot \text{unit}^{\times}) \cdot \text{cross} =$$

(by lemma 4.5:1 for $L_1 \otimes \ldots \otimes L_k$)

$$\lambda(l_1, \ldots, l_k).\bigsqcup\{(\text{cross}^{\boxtimes} \cdot \text{unit}^{\times})(\text{cross}(ib_1, \ldots, ib_k)) \mid$$
$$\forall j: ib_j \in IB_{L_j} \wedge \text{cross}(ib_1, \ldots, ib_k) \sqsubseteq \text{cross}(l_1, \ldots, l_k)\} =$$

(by lemma 3.2:7 and $\text{cross}^{\boxtimes} \cdot \text{unit}^{\times} \cdot \text{cross} = \text{cross}$)

$$\lambda(l_1, \ldots, l_k).\bigsqcup\{\text{cross}(ib_1, \ldots, ib_k) \mid$$
$$\forall j: (ib_j \in IB_{L_j} \wedge ib_j \sqsubseteq l_j)\} =$$

(by cross seperately additive hence seperately irreducibly generated)

$$\text{id} \cdot \text{cross}$$

As in the previous paragraph it follows that id = lin(cross$^{\boxtimes}$·unit$^{\prime}$) and then id $\sqsubseteq$ cross$^{\boxtimes}$·unit$^{\prime}$ is immediate. ///

We shall not consider the corresponding development for $\otimes$. One technical problem in doing so is that $L_1 * \ldots * L_k$ needs not be an $\underline{\underline{\text{ACLI}}}$ object even if all L_i are. (In fact $L_2 * L_2$ in the proof of lemma 4.4:3 is an example.) To overcome these problems one might adopt the suggestion in the final remark in section 4.4 (but this would make $\otimes$ uninteresting).

5 STRONG ABSTRACT INTERPRETATION

The development of abstract interpretation in the previous chapters was based on the relational powerdomain and approximations to it. This setting is useful for many purposes but there seems to be situations where the relational powerdomain excludes subsets that are of interest. This motivates a development based on a powerdomain that allows more subsets to be considered.

The motivation for why more subsets seem to be needed arises from program transformations whose safeness depend on whether or not the execution of a certain piece of program terminates. As an example consider an imperative programming language and the statements

(1) $\underline{\text{let}}$ p($\underline{\text{value}}$ x) $\underline{\text{be}}$ b_1 $\underline{\text{in}}$ b_2

and

(2) $\underline{\text{let}}$ p($\underline{\text{name}}$ x) $\underline{\text{be}}$ b_1 $\underline{\text{in}}$ b_2

Both statements declare a procedure p with one argument x. The parameter mechanism is call-by-value in (1) and call-by-name in (2). The procedure p may be called in b_2 and is defined by b_1. We shall further assume that x does not occur in b_1 (or at least that there is a way of executing b_1 such that x is not referenced). A similar set-up may be imagined for an applicative programming language where b_1 and b_2 will be expressions.

One program transformation might be to replace (2) by (1). For this to be safe every argument e to a call of p in b_2 must terminate upon execution. For otherwise (2) may terminate in a situation where (1) does not and therefore the program transformation does not preserve the overall meaning of the program. This transformation was considered in /Myc80,Myc81/ in the context of optimising applicative programs. Another program transformation is to replace (1) by (2). This is essentially what happens in an applicative language when expanding a call in-line. To see this note that (2) is equivalent to

$$(2')\quad b_2\left[\, b_1[e/x] \;/\; p(e)\right]$$

that is b_2 with all occurrences of a call p(e) replaced by the body b_1 except that x is replaced by e. In an imperative programming language in-line expansion resembles replacing (2) by (1). For (1) is essentially equivalent to

$$(1')\quad b_2\left[\, x:=e;b_1 \;/\; p(e)\right]$$

We shall not go further into such transformations, but the examples show that a data flow analysis for detecting safe approximations to "will terminate" may be useful. Therefore it ought to be possible to prove it correct within the framework of abstract interpretation.

It is natural to take the point of view that possible non-termination of an expression is just another facet of which value it may evaluate to. This is in line with domain theory where the value $\perp$ is returned by a function iff it does not terminate. This means that the denotation $\underline{C}[\![e]\!]...$ of e in the collecting semantics should not be <u>forced</u> to contain $\perp$ as is the case with the relational powerdomain. In section 3.1 it was argued that the Smyth powerdomain is not suitable (only one infinite set) and this also holds for the Plotkin

powerdomain (all infinite sets are forced to contain $\perp$). Therefore a powerdomain with fewer restrictions is studied in section 5.2. Unfortunately continuity properties may fail and we rely on monotonicity instead. The relevant domain theory is covered in section 5.1. (Throughout this chapter recursive domains will be excluded from consideration.)

Section 5.3 gives the standard and collecting semantics for the applicative language of section 2.4. They are shown to be related much as in section 3.3 but the collecting semantics is a more faithful representation of the standard semantics than before (because the powerdomain allows more subsets to be considered). In section 5.4 it is argued that abstract interpretation should use spaces with two partial orders. For the powerdomain they are the Egli-Milner order (used for least fixed points) and subset inclusion (used for safe approximation). These partial orders are the same for the relational powerdomain so this is a more general outlook than in the previous chapters. When the abstraction and concretization functions preserve the relations in a suitable way a development of abstract interpretation can be given. The major application in section 5.5 is to validate two data flow analyses taken from /Myc81/ and there used to detect when call-by-name can be replaced by call-by-value. One of the analyses seems to require this stronger approach.

Section 5.6 defines a small nondeterministic programming language and its nondeterministic semantics. Also a deterministic semantics is given where oracles (strings of 0's and 1's) are used to resolve which choice is to be made. Abstract interpretation is then extended so as to allow some arguments not to be "collected"; for the collecting semantics this means we may have functionalities $N_\perp \times \mathcal{P}(N_\perp) \to \mathcal{P}(N_\perp)$.

When applying this to the deterministic semantics it is shown that the nondeterministic semantics results. This gives a clear perspective upon what nondeterminism "is". It is a natural data flow analysis question to investigate what happens if not all oracles are possible and this is done in section 5.7. Under reasonable assumptions it is shown that this gives a semantics $\underline{D}_O$ that specifies convergence more often than the nondeterministic semantics. However, if the non-deterministic semantics is modified to use another fixed point operator it is possible to obtain a semantics that specifies convergence more often that $\underline{D}_O$. In this way $\underline{D}_O$ is bounded by the two nondeterministic semantics.

The main purpose of this chapter is to explore a theory of abstract interpretation where aspects of termination may be dealt with. Much work remains in order to attain the same degree of "universality" that holds for the development of the previous chapters. Even then there seeems to be applications where the "more general outlook" of this chapter is too narrow as will be discussed in chapter 6.

5.1 NON-CONTINUOUS DOMAIN THEORY

In this section we overview a theory of domains where assumptions about continuity are replaced by assumptions about monotonicity. Most of the concepts are quite standard and can be found in /ApP182/ or /Mar76/.

Let $D = (D,\sqsubseteq)$ be a partially ordered set. A subset Y of D is directed iff for every finite subset Y' of Y there is an element y of Y that is an upper bound for Y', i.e. $\forall y' \in Y' : y' \sqsubseteq y$. A directed set cannot be empty because Y' can be chosen as the empty set. The

partially ordered set is directed complete (is a dcpo) iff it has a least element $\perp$ and for every directed subset Y of D there is a least upper bound $\bigsqcup Y$ in D. It follows from section 2.2 that a dcpo is a cpo and an algebraic cpo is a dcpo. The definition of dcpo will be further clarified shortly (equivalent definitions and different concepts). For the present it suffices to state that not all cpo's are dcpo's.

...

Only a few constructions upon dcpo's will be needed. For a set S the dcpo $S_\perp$ has as elements $S \cup \{\perp\}$ (assuming $\perp$ is not an element of S) and is partially ordered by $x \sqsubseteq y$ iff $x = \perp$ or $x = y$. If $k > 1$ and D_i are dcpo's then so is the cartesian product $D_1 \times \ldots \times D_k$. It is partially ordered componentwise and the least upper bound is also componentwise, i.e.

$$\bigsqcup Y = (\bigsqcup \{y\downarrow_1 \mid y \in Y\}, \ \ldots \ , \bigsqcup \{y\downarrow_k \mid y \in Y\})$$

Also the smash product

$$D_1 * \ldots * D_k = \{\text{smash}(d_1, \ldots, d_k) \mid \forall i: \ d_i \in D_i\}$$

is a dcpo with least upper bounds determined componentwise. For dcpo's D and E the function space $D \to E$ is redefined to consist of all monotonic functions from D to E. It is partially ordered pointwise and gives a dcpo where least upper bounds are pointwise, i.e. $(\bigsqcup Y)(d) = \bigsqcup \{f(d) \mid f \in Y\}$.

Every monotonic function over a dcpo has a least fixed point. The construction and proof uses ordinal numbers and transfinite induction (see /Hal60/). Let $f: D \to D$ be a monotonic function and D a dcpo. Inductively define, for each ordinal λ, the element $f^{(\lambda)}$ of D by

$$f^{(\lambda)} = f(\bigsqcup\{f^{(K)} \mid K<\lambda\})$$

This obviously makes sense when λ is a natural number and gives $f^{(n)} = f^{n+1}(\bot)$. To see it makes sense in general it suffices to show that $\lambda_1<\lambda_2<\lambda$ implies $f^{(\lambda_1)} \sqsubseteq f^{(\lambda_2)}$ as then $\{f^{(K)} \mid K<\lambda\}$ is directed. But this follows from the monotonicity of f and

$$\{f^{(K)} \mid K<\lambda_1\} \subseteq \{f^{(K)} \mid K<\lambda_2\}$$

Next define

$$LFP(f) = f^{(\lambda_D)}$$

for λ_D the (ordinal number corresponding to the) cardinality of the powerset of D.

Lemma 5.1:1. LFP(f) is the least fixed point of f. ///

Proof We first show that LFP(f) is less than or equal to any fixed point. So let $f(d) \sqsubseteq d$ and show by transfinite induction on $\lambda \leq \lambda_D$ that $f^{(\lambda)} \sqsubseteq d$. If the result holds for all ordinals $K<\lambda$ then

$$\bigsqcup\{f^{(K)} \mid K<\lambda\} \sqsubseteq d$$

and monotoniciy of f and the assumption about d gives the result for λ too.

We next show that LFP(f) is a fixed point. It cannot be that all $f^{(\lambda)}$ for $\lambda<\lambda_D$ are distinct as this would contradict Cantor's theorem /Hal60/. So let λ_f be the least ordinal λ such that $f^{(\lambda)} = f^{(K)}$ for some ordinal K that is not λ. This means that there is an ordinal $K_f > \lambda_f$ such that $f^{(\lambda_f)} = f^{(K_f)}$. We have $\lambda_f < \lambda_f+1 \leq K_f < \lambda_D$ and because $f^{(\lambda)}$ depends monotonically on λ, and because $f^{(\lambda+1)} = f(f^{(\lambda)})$, this gives

$$f^{(\lambda_f)} \sqsubseteq f(f^{(\lambda_f)}) \sqsubseteq f^{(K_f)} \sqsubseteq LFP(f)$$

But by choice of κ_f this shows that $f^{(\lambda_f)}$ is a fixed point of f that is less than or equal to LFP(f). By the first half of the proof it must equal LFP(f). ///

Corresponding to the definition of least fixed points there is the following rule of fixed point induction. A predicate Q on D is (directed) admissible iff $Q(\bot)$ holds and $Q(\bigsqcup Y)$ holds for every directed set Y such that $\forall y \in Y$: Q(y). The fixed point induction principle says

$$\frac{\forall d \in D:\ Q(d) \Rightarrow Q(f(d))}{Q(LFP(f))}$$

for every directed admissible predicate Q and monotonic function f over a dcpo D. The proof of this principle is straightforward by transfinite induction. (For ordinal λ one shows $Q(f^{(\lambda)})$ using that $Q(f^{(\kappa)})$ holds when $\kappa < \lambda$.)

If Q is a predicate on $D \to E$ we write $D \to_Q E$ for the subset of those monotonic functions that satisfy Q. This gives a partially ordered set when using the partial order of $D \to E$. The predicate Q is admissible iff $D \to_Q E$ is a dcpo with the same least element as in $D \to E$ and with the same least upper bounds of directed sets as in $D \to E$. Strictness is an admissible predicate and so is directed continuity (which means that least upper bounds of directed sets are preserved as in $f(\bigsqcup Y) = \bigsqcup\{f(y) \mid y \in Y\}$).

Clarifying the definition of dcpo

To clarify the definition of dcpo we need the following concepts. A directed chain is a directed set that is totally ordered. Recall that an ordinal λ is partially ordered by $\leq$ /Hal60/. A chain

indexed by λ is a monotonic function from λ (into some partially ordered set D). A chain indexed by ω, the smallest infinite ordinal /Hal60/, then is what was called a chain in chapter 1.

A chain $(d_\kappa)_\kappa$ indexed by λ is cofinal in the directed set Y iff each d_κ is an element of Y and for each $y \in Y$ there is an ordinal $\kappa < \lambda$ such that $y \sqsubseteq d_\kappa$. If $(d_\kappa)_{\kappa<\lambda}$ is cofinal in Y then Y and $\{d_\kappa \mid \kappa<\lambda\}$ have the same upper bounds. So $\bigsqcup Y$ exists iff $\bigsqcup\{d_\kappa \mid \kappa<\lambda\}$ exists and they are equal if one exists.

Lemma 5.1:2. For every directed chain Y there is a cofinal chain $(d_\kappa)_\kappa$ indexed by some cardinal $\lambda \leq \mathrm{card}(Y)$. ///

Proof We first show the weaker result where the assumption that λ is a cardinal is replaced by the assumption that λ is merely an ordinal. Since card(Y) is a cardinal there is a bijection $i:\mathrm{card}(Y) \to Y$. Let g be a choice function /Hal60/ for the set Y. The definition of d_κ and λ is by transfinite induction on $\lambda' < \mathrm{card}(Y)$.

If $\{d_\kappa \mid \kappa<\lambda'\}$ does not have an upper bound in Y we stop the construction and set $\lambda = \lambda'$. Otherwise let $d_{\lambda'}$ be the larger of $i(\lambda')$ and some upper bound of $\{d_\kappa \mid \kappa<\lambda'\}$. (The upper bound can be chosen uniquely by using the choice function g.) If d_κ are defined for all $\kappa<\mathrm{card}(Y)$ we set $\lambda=\mathrm{card}(Y)$.

Clearly $(d_\kappa)_{\kappa<\lambda}$ is a chain indexed by λ and each d_κ is an element of Y. To see it is cofinal fix $y \in Y$. If $i^{-1}(y)<\lambda$ it is obvious that $y \sqsubseteq d_{i^{-1}(y)}$. If $i^{-1}(y) \geq \lambda$ we must have that y is not an upper bound of $\{d_\kappa \mid \kappa<\lambda\}$. Hence there is $\kappa<\lambda$ such that $\neg(d_\kappa \sqsubseteq y)$ and this implies $y \sqsubseteq d_\kappa$ because Y is totally ordered.

To prove the result stated in the lemma let λ be the smallest ordinal such that there is a cofinal chain $(d_\kappa)_{\kappa<\lambda}$ in Y. Next use the weak result above on (Y=) $\{d_\kappa|\kappa<\lambda\}$ to get a cofinal chain $(d'_\kappa)_{\kappa<\lambda'}$ with $\lambda'\leq \mathrm{card}(\lambda)\leq\lambda$. Since $(d'_\kappa)_{\kappa<\lambda'}$ is cofinal in Y too this shows that $\lambda'=\lambda$. Hence $\lambda=\mathrm{card}(\lambda)$ and therefore λ is a cardinal. ///

It follows from example 1 in /Mar76/ that the lemma may fail if Y is only a directed set.

The next theorem shows that it was not important that the definition of dcpo focused on directed sets.

Theorem 5.1:3. Let λ be a cardinal and D a partially ordered set with a least element. Consider the following assumptions about which least upper bounds should exist in D:

(1) of all directed sets (of cardinality at most λ)

(2) of all directed chains (of cardinality at most λ)

(3) of all chains indexed by an ordinal (at most λ)

(4) of all chains indexed by a cardinal (at most λ)

All these statements are equivalent and this also holds if the parts in parentheses are ignored. ///

Proof Let (1'),...,(4') be (1),...,(4) with the parts in parentheses ignored. It is obvious that (1) $\Rightarrow$ (2) $\Rightarrow$ (3) $\Rightarrow$ (4) and that (1') $\Rightarrow$ (2') $\Rightarrow$ (3') $\Rightarrow$ (4'). That (2) $\Rightarrow$ (1) and (2') $\Rightarrow$ (1') is by corollaries 1 and 2 in /Mar76/. (The proof essentially proceeds by transfinite induction on the cardinality of the directed set. For each directed set the assumption (2) or (2') is used to construct a directed chain with the same least upper bound.) That (4) $\Rightarrow$ (2) is

by the previous lemma and (4')⇒(2') is similar. ///

What was important in the definition of dcpo is that there was no cardinality restrictions upon the directed sets considered. Let a λ-cpo mean a partially ordered set with a least element and with least upper bounds of directed sets of cardinality at most λ. Then D is a dcpo iff it is a card(D)-cpo and cpo in the sense of section 2.2 means ω-cpo. ...

Fact 5.1:4. If $\lambda_1 < \lambda_2$ then every λ_2-cpo is a λ_1-cpo. ///

Fact 5.1:5. If $\lambda_1 < \lambda_2$ are infinite cardinals there is a λ_1-cpo that is not a λ_2-cpo. ///

Proof Let λ_3 be the smallest cardinal strictly greater than λ_1. Since it is a limit ordinal it has no greatest element and therefore the directed subset λ_3 has no least upper bound. This shows that λ_3 is not a λ_2-cpo. To see it is a λ_1-cpo let y be a directed subset of cardinality at most λ_1. The elements of y have cardinality at most λ_1 and therefore so has $\bigcup y$ (because $\lambda_1 \cdot \lambda_1 = \lambda_1$ /Hal60 p.97/). So $\bigcup y$ is an element of λ_3 and is the least upper bound /Hal60 p.79/. ///

If n is a natural number then n-cpo means a partially ordered set with a least element. The partially ordered sets considered in /Plo82/ are the Ω-cpo's, where Ω is the smallest non-countable ordinal /Hal60/.

An advantage of not placing cardinality restrictions upon the directed sets is that it is not necessary with continuity assumptions in order to guarantee that least fixed points exist. Another consequence is:

Fact 5.1:6. A dcpo is a complete lattice iff it is a semi-lattice. ///

This is corollary 5 in /Mar76/. The idea in "if" is to consider a nonempty set Y. Let i:card(Y)→Y be a bijection and define d_λ by $i(\lambda) \sqcup \bigsqcup\{d_\kappa \mid \kappa<\lambda\}$. Then $\bigsqcup Y$ exists and equals $\bigsqcup\{d_\kappa \mid \kappa<\mathrm{card}(Y)\}$.

The definition of directed continuous can be analysed in much the same way. For a monotonic function between dcpo's one can view (1) to (4) above as assumptions about which least upper bounds it should preserve. One can show that an analogue of theorem 5.1:3 holds. (For (2)⇒(1) use corollary 3 of /Mar76/.) Finally a monotonic function is completely additive iff it is strict, additive and directed continuous.

5.2 POWERDOMAINS

As has been argued in the introduction to this chapter the Plotkin powerdomain /Plo76/ does not allow (to distinguish between) all the sets that are desired. In this section a preliminary theory is given for a powerdomain that is better in this respect.

The powerdomain of a dcpo D has as elements certain subsets of D. A subset Y of D is <u>convex</u> iff whenever $d_1 \sqsubseteq d_2 \sqsubseteq d_3$ with d_1 and d_3 elements of Y then so is d_2. The <u>Egli-Milner order</u> $\sqsubseteq_{EM}$ is defined between subsets by

$$Y_1 \sqsubseteq_{EM} Y_2 \text{ iff } \forall y_1 \in Y_1: \exists y_2 \in Y_2: \; y_1 \sqsubseteq y_2$$
$$\text{and } \forall y_2 \in Y_2: \exists y_1 \in Y_1: \; y_1 \sqsubseteq y_2$$

<u>Definition</u> The powerdomain $\mathcal{P}(D)$ of a dcpo D is defined as $(\{Y \subseteq D \mid Y \text{ is convex and nonempty}\}, \sqsubseteq_{EM})$. ///

It is straightforward to verify that this gives a partially ordered set with $\{\perp\}$ as least element. It is not always a dcpo and we

now investigate when it is.

As a dcpo

It is convenient to define, for every subset Y of D, the sets

$$LC(Y) = \{d \in D \mid \exists y \in Y: d \sqsubseteq y\} \qquad \text{(left closure)}$$

$$RC(Y) = \{d \in D \mid \exists y \in Y: d \sqsupseteq y\} \qquad \text{(right closure)}$$

$$CON(Y) = LC(Y) \cap RC(Y) \qquad \text{(convex closure)}$$

$$MAX(Y) = \{y \in Y \mid \forall y' \in Y: y' \sqsupseteq y \Rightarrow y' = y\} \qquad \text{(maximal elements)}$$

One may then reformulate $Y_1 \sqsubseteq_{EM} Y_2$ as

$$Y_1 \subseteq LC(Y_2) \quad \text{and} \quad RC(Y_1) \supseteq Y_2$$

and a set Y is convex iff Y = CON(Y). A dcpo is of finite height (is a fdcpo) iff every directed subset contains its own least upper bound.

Fact 5.2:1. If D is of finite height then $Y \subseteq LC(MAX(Y))$, i.e. for all $y \in Y$ there exists $y_M \in MAX(Y)$ such that $y \sqsubseteq y_M$. ///

Proof For the sake of contradiction suppose that the fact is violated when $y = y_0$. Construct y_{n+1} as some element of Y that is strictly greater than y_n. Then $\{y_n \mid n \geq 0\}$ is a directed set that does not contain its least upper bound. This is the desired contradiction with the finite height of D. ///

Two elements d_1 and d_2 are incomparable iff neither $d_1 \sqsubseteq d_2$ nor $d_1 \sqsupseteq d_2$. The elements of a set Y are pairwise incomparable iff every element of Y is incomparable with every other element of Y. This is the case iff Y = MAX(Y).

The following definition is central for investigating when $\mathcal{P}(D)$ is a dcpo.

Definition A dcpo D is benign iff for every subset Y_L and Y_R of D and element z of D such that

(1) Y_L is infinite

(2) $Y_L = MAX(Y_L)$, i.e. the elements of Y_L are pairwise incomp.

(3) $Y_L \subseteq LC(\{z\})$, i.e. each element of Y_L is "below" z

(4) $Y_L \subseteq L\dot{C}(Y_R)$

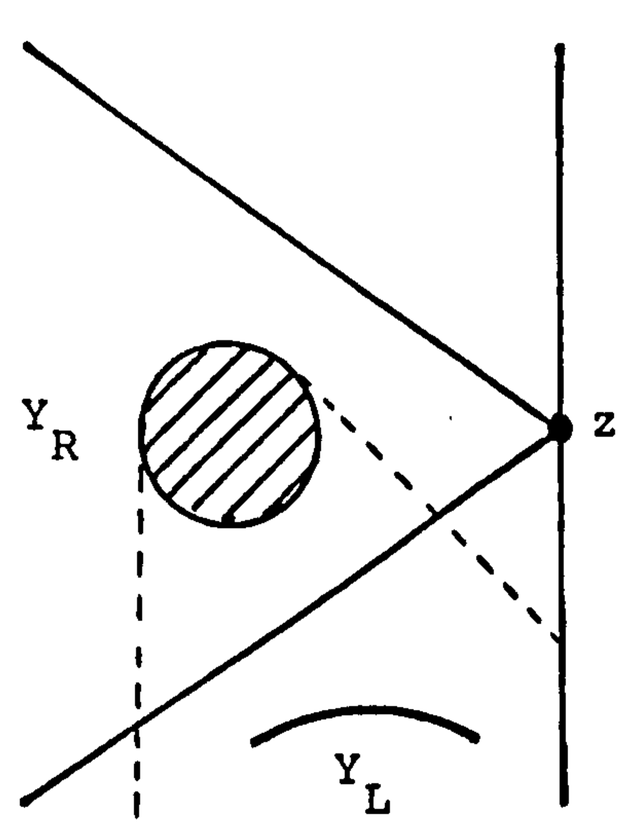
(5) each element of Y_R is incomparable with z

it is also the case that

(6) $\exists y \in LC(Y_R) \cap LC(\{z\}) \cap (RC(Y_L) - Y_L)$

(where - denotes set difference). ///

Conditions (1) to (5) are illustrated by



It is hard to give an "intuitive" motivation for this concept but we have:

Theorem 5.2:2. Let D be a dcpo of finite height.

(i) $\mathcal{P}(D)$ is a dcpo iff D is benign

(ii) $\bigsqcup y = LC(\bigcup y) \cap \bigcap_{Y \in y} RC(Y)$ when D is benign

(iii) the class of fdcpo's such that

no infinite set of pairwise incomparable elements has an upper bound

is the largest class of benign fdcpo's that is closed under cartesian product. ///

Proof of "only if" in (i): Suppose by way of contradiction that $\mathcal{E}(D)$ is a dcpo but that D is a fdcpo that is not benign. Let Y_L, Y_R and z fulfil (1) to (5) and violate (6). We shall construct a directed chain in $\mathcal{E}(D)$ and obtain the desired contradiction.

The cardinal number $\beta=\mathrm{card}(Y_L)$ is infinite by (1). Let $i:\beta\to Y_L$ be a bijection and define for $\lambda<\beta$ the set

$$Y_\lambda = \mathrm{CON}(Y_R \cup \{i(\kappa) \mid \lambda\leq\kappa<\beta\})$$

To see that this gives a chain indexed by β let $\alpha<\lambda<\beta$. Since $Y_\alpha \supseteq Y_\lambda$ we get $\mathrm{RC}(Y_\alpha)\supseteq Y_L$. Also $\mathrm{MAX}(Y_\alpha) = \mathrm{MAX}(Y_R) = \mathrm{MAX}(Y_\lambda)$ and the previous fact shows that $Y_\lambda \subseteq \mathrm{LC}(\mathrm{MAX}(Y_\alpha)) \subseteq \mathrm{LC}(Y_\lambda)$. It follows that $Y_\alpha \sqsubseteq_{EM} Y_\lambda$.

Both $M_1 = \mathrm{MAX}(Y_R)$ and $M_2 = \mathrm{MAX}(Y_R)\cup\{z\}$ are convex, nonempty sets and we now show that they are upper bounds of $(Y_\lambda)_{\lambda<\beta}$. That $Y_\lambda\subseteq\mathrm{LC}(M_i)$ is immediate for both i=1 and i=2. Next that $\mathrm{RC}(Y_\lambda)\supseteq M_1$ is immediate and that $\mathrm{RC}(Y_\lambda)\supseteq M_2$ is because each Y_λ contains an element of Y_L and z will be greater than or equal to that element.

Because $\mathcal{E}(D)$ is a dcpo there is a least upper bound M of $(Y_\lambda)_{\lambda<\beta}$ and we have $M \sqsubseteq_{EM} M_1$ and $M \sqsubseteq_{EM} M_2$. This gives $m\in M$ such that $m\subseteq z$ and $y\in Y_R$ such that $m\subseteq y$. It follows that $m\in \mathrm{LC}(Y_R)\cap\mathrm{LC}(\{z\})$. Also for each λ there is $y_\lambda\in Y_L\cap Y_\lambda$ such that $y_\lambda\subseteq m$. This shows that $m\in\mathrm{RC}(Y_L)$ and because (6) is violated we must have $m\in Y_L$. But for

$i^{-1}(m)<\lambda<\beta$ the existence of y_λ contradicts assumption (2).

Proof of "if" in (i) and of (ii): Suppose that D is a benign fdcpo with $\mathcal{Y}$ as a directed subset. Let UB be defined by the formula in (ii).

We first show that UB is a convex, nonempty set that is an upper bound of $\mathcal{Y}$. To see it is convex let $d_1 \sqsubseteq d_2 \sqsubseteq d_3$ with $d_1 \in UB$ and $d_3 \in UB$. Because $d_3 \in LC(\cup\mathcal{Y})$ we get $d_2 \in LC(\cup\mathcal{Y})$ and because $d_1 \in \cap_{Y\in\mathcal{Y}} RC(Y)$ we get $d_2 \in \cap_{Y\in\mathcal{Y}} RC(Y)$. This shows that $d_2 \in UB$. To see that it is not empty we note that $MAX(\cup\mathcal{Y})$ is not empty and show that $MAX(\cup\mathcal{Y})$ is a subset of UB. So let $y \in MAX(\cup\mathcal{Y})$ and note that this gives $Y \in \mathcal{Y}$ such that $y \in Y$. To see that $y \in UB$ it suffices to consider $Y' \in \mathcal{Y}$ and show $y \in RC(Y')$. Let Y" be an upper bound in $\mathcal{Y}$ of Y and Y'. By the assumption about $\mathcal{Y}$ we have $Y \sqsubseteq_{EM} Y"$ and $Y' \sqsubseteq_{EM} Y"$. This shows that $y \in Y"$ and $y \in RC(Y)$. Finally, to see that it is an upper bound consider $Y \in \mathcal{Y}$ and show $Y \sqsubseteq_{EM} UB$. For $y \in Y$ we use the previous fact to find $y_M \in MAX(\cup\mathcal{Y})$ such that $y \sqsubseteq y_M$ and by the above $y_M \in UB$. For $y' \in UB$ we have $y' \in RC(Y)$ and therefore there is $y \in Y$ such that $y \sqsubseteq y'$.

Next let M be another upper bound of $\mathcal{Y}$ and show $UB \sqsubseteq_{EM} M$. Note that UB is a subset of $\cup\mathcal{Y}$. For $y \in UB$ we therefore have $Y \in \mathcal{Y}$ such that $y \in Y \sqsubseteq_{EM} M$ and this gives $z \in M$ such that $y \sqsubseteq z$. Next let $z \in M$ and find $y \in UB$ such that $y \sqsubseteq z$. For the sake of contradiction assume that no such y exists. Then also there is no $y \in UB$ such that $z \sqsubseteq y$ for otherwise z would be an element of UB thereby contradicting the assumption.

Define $Y_R = MAX(UB)$ and $Y_{VL} = MAX((\bigcup \mathcal{Y}) \cap LC(\{z\}))$ and note that both sets are convex and not empty (for Y_{VL} use that M is an upper bound of $\mathcal{Y}$). Define $Y_L = MAX(LC(Y_R) \cap LC(\{z\}) \cap RC(Y_{VL}))$ and note that this gives a convex and nonempty set (the latter is because Y_{VL} is a subset of the set of which MAX is taken).

Suppose that Y_L is infinite. Clearly conditions (1) to (5) of benign hold. This gives by (6) a certain element y. Since $y \in RC(Y_L) - Y_L$ we get $y \in RC(Y_{VL})$ and because $y \in LC(\{z\}) \cap LC(Y_R)$ this gives $y \in LC(Y_L)$. But this gives a contradiction with $y \in RC(Y_L) - Y_L$. Next suppose that Y_L is finite and that $y_1, \ldots, y_n$ are all the elements. Each y_i is in $LC(Y_R)$ and hence in $LC(\bigcup \mathcal{Y})$. Since $y_i \sqsubseteq z$ the assumption about "no such y" gives $Y_i \in \mathcal{Y}$ such that $y_i \notin RC(Y_i)$. Let Y be an upper bound in $\mathcal{Y}$ of $Y_1, \ldots, Y_n$. Then $y_i \notin RC(Y)$ must be the case for all i. But $Y \sqsubseteq_{EM} M$ gives $y' \in Y$ such that $y' \sqsubseteq z$. Therefore there is $y'' \in Y_{VL}$ such that $y' \sqsubseteq y''$ and $y_i \in Y_L$ such that $y' \sqsubseteq y_i$. This shows that $y_i \in RC(Y)$ and is the desired contradiction.

Proof of (iii): We first show that the class described is a subclass of the benign fdcpo's and is closed under cartesian product. Clearly each fdcpo in the class is benign. If $D_1, \ldots, D_k$ $(k>1)$ are fdcpo's in the class then $D_1 \times \ldots \times D_k$ is a fdcpo. To see it is in the class it suffices to consider k=2 as $D_1 \times \ldots \times D_k$ is isomorphic to $((D_1 \times D_2) \times \ldots \times D_k)$.

Suppose by way of contradiction that Y is an infinite subset of $D_1 \times D_2$ that has an upper bound (u_1, u_2) and whose elements are pairwise incomparable. Both $\{y\downarrow 1 \mid y \in Y\}$ and $\{y\downarrow 2 \mid y \in Y\}$ have an upper bound and at least one is infinite. Suppose (without loss of generality) it is $\{y\downarrow 1 \mid y \in Y\}$ and call this set L_0. We construct a strictly

decreasing sequence $(d_n)_{n\geq 1}$ of elements of L_0 and a strictly decreasing sequence $(L_n)_n$ of infinite subsets of L_0. Since L_{n-1} is infinite and $MAX(L_{n-1})$ is finite there is an element of $MAX(L_{n-1})$ such that infinitely many elements of L_{n-1} is less than it. Let d_n be such an element of $MAX(L_{n-1})$ and put $L_n = \{d \in L_{n-1} \mid d \sqsubseteq d_n \wedge d \neq d_n\}$. Clearly $n > m$ implies that $d_n \sqsubseteq d_m$ and $d_n \neq d_m$.

It is immediate that there exists e_n such that $(d_n, e_n) \in Y$ for all n. The sequence $(e_n)_n$ must be non-decreasing, i.e. $m > n$ implies $\neg(e_m \sqsubseteq e_n)$, because the elements of Y are pairwise incomparable. Therefore, if $m > n$ then $e_m \neq e_n$ and if e_m and e_n are comparable we must have $e_m \sqsupseteq e_n$. Next fix n_0. It cannot be that for each $n \geq n_0$ there is $m > n$ such that e_m and e_n are comparable. For if this is the case we can construct an infinite strictly increasing sequence and this contradicts the finite height of D_2. So define $f(n_0)$ as the least $n_1 > n_0$ such that $m > n_1$ implies that e_m and e_{n_1} are incomparable. Then $\{e_j \mid \exists i: j = f^i(0)\}$ is an infinite set of pairwise incomparable elements. Since it has u_2 as an upper bound this contradicts the assumption about D_2 being in the class.

To see that the described class is largest suppose by way of contradiction that it is not. So there is another class containing a benign fdcpo D that is not in the described class. Hence $D \times D$ is a benign fdcpo and there is an infinite subset $\{d_n \mid n \geq 1\}$ of pairwise incomparable elements of D with an upper bound u. (It is no restriction to assume that the set is countable.) Because D is of finite height there exists a maximal lower bound l for d_1 and d_2. In $D \times D$ define $Y_L = \{(d_n, l) \mid n \geq 1\}$, $Y_R = \{(d_n, d_1) \mid n \geq 1\}$ and $z = (u, d_2)$. Conditions (1) to (5) of benign hold so by (6) there is an element y and numbers n and m such that $y \sqsubseteq (u, d_2)$ and $y \sqsubseteq (d_n, d_1)$ and $y \sqsupseteq (d_m, l)$

and $y \neq (d_m, 1)$. The first three statements show that $y = (d_m, 1)$ and this contradicts the fourth. Hence the described class is largest.

///

As an augmented dcpo

So far the powerdomain has been studied with respect to the Egli-Milner order $\sqsubseteq_{EM}$. Another partial order that may be studied is subset inclusion $\subseteq$. We shall see in section 5.4 that both are of importance for abstract interpretation. (The Egli-Milner order will be used to define least fixed points and subset inclusion to express safe approximation.) Analogously to /Egl75/ and /HeAs80/ this motivates a study of structures with two partial orders.

An augmented dcpo $B = (B, \sqsubseteq, \subseteq)$ is a dcpo $(B, \sqsubseteq)$ such that $(B, \subseteq)$ is a partially ordered set. The augmentation is admissible iff $\subseteq$ is an admissible predicate upon $B \times B$ (with respect to $\subseteq$ defined componentwise). The augmentation is complete iff $(B, \subseteq)$ has least upper bounds of all nonempty subsets and the least upper bound operator $\cup$ satisfies the following monotonicity property:

> if Y and Y' are nonempty subsets of B and $Y \sqsubseteq_{EM} Y'$
> then $\cup Y \subseteq \cup Y'$

(Here $\sqsubseteq_{EM}$ is defined with respect to $\subseteq$, i.e. $Y \sqsubseteq_{EM} Y'$ iff for all $Y' \in Y'$ there is $Y \in Y$ such that $Y \sqsubseteq Y'$ and ... .) The following fact is used in the proof of the theorem below and gives a connection between an admissible augmentation and what is sometimes called a continuous semi-lattice /PloLN/.

Fact 5.2:3. If $B = (B, \sqsubseteq, \subseteq)$ is a completely augmented dcpo and $\upsilon = \lambda(Y_1, Y_2).\cup\{Y_1, Y_2\}$ is directed continuous then the augmentation

is admissible. ///

Proof Clearly $\perp \sqsubseteq_\perp \perp$ holds and let $\mathcal{Y}$ be a directed subset of $B^{\times}B$ such that $Y_1 \sqsubseteq Y_2$ for every $(Y_1,Y_2) \in \mathcal{Y}$. Then $\upsilon(Y_1,Y_2) = Y_2$ holds for $(Y_1,Y_2) \in \mathcal{Y}$ and the calculation

$$(\sqcup\mathcal{Y})\downarrow 2 =$$
$$\sqcup\{Y_2 \mid (Y_1,Y_2) \in \mathcal{Y}\} =$$
$$\sqcup\{\upsilon(Y_1,Y_2) \mid (Y_1,Y_2) \in \mathcal{Y}\} =$$
$$\upsilon(\sqcup\mathcal{Y})$$

shows that $(\sqcup\mathcal{Y})\downarrow 1 \sqsubseteq (\sqcup\mathcal{Y})\downarrow 2$ as was to be shown. ///

The definition of augmented dcpo pays off because the powerdomain is such a structure.

Theorem 5.2:4. If D is a benign fdcpo then $\mathcal{E}(D) = (\mathcal{E}(D), \sqsubseteq_{EM}, \sqsubseteq)$ is an admissibly and completely augmented dcpo with $\upsilon\mathcal{Y} = \text{CON}(\sqcup\mathcal{Y})$. ///

Proof Clearly $\sqsubseteq$ is a partial order upon $\mathcal{E}(D)$. If $\mathcal{Y}$ is a set of elements of $\mathcal{E}(D)$ then an element Y of $\mathcal{E}(D)$ is an upper bound iff $\sqcup\mathcal{Y} \sqsubseteq Y$. Since Y is convex this holds iff $\text{CON}(\sqcup\mathcal{Y}) \sqsubseteq Y$. This shows that $(\mathcal{E}(D), \sqsubseteq)$ has least upper bounds of nonempty subsets and that the least upper bound is given by υ.

If $\mathcal{Y} \sqsubseteq_{EM} \mathcal{Y}'$ are nonempty subsets of $\mathcal{E}(D)$ we must show $\upsilon\mathcal{Y} \sqsubseteq_{EM} \upsilon\mathcal{Y}'$. Since $\text{CON}(Y) \sqsubseteq_{EM} Y \sqsubseteq_{EM} \text{CON}(Y)$ holds for all subsets Y of D it suffices to show that $\sqcup\mathcal{Y} \sqsubseteq_{EM} \sqcup\mathcal{Y}'$. When $y \in \sqcup\mathcal{Y}$ there is $Y \in \mathcal{Y}$ such that $y \in Y$. Hence there is $Y' \in \mathcal{Y}'$ with $Y \sqsubseteq_{EM} Y'$ and this gives $y' \in Y'$ such that $y \sqsubseteq y'$. Clearly y' is an element of $\sqcup\mathcal{Y}'$. When $y' \in \sqcup\mathcal{Y}'$ one finds $y \in \sqcup\mathcal{Y}$ such that $y \sqsubseteq y'$ in a similar way.

To see that $\sqsubseteq$ is admissible we use the previous fact and show that $\upsilon = \lambda(Y_1,Y_2).\sqcup\{Y_1,Y_2\}$ is directed continuous. It suffices to show that υ is continuous in each argument and because υ is commutative it suffices to consider the left argument. Let Z be an element of $\mathcal{E}(D)$ and $\mathcal{Y}$ a directed subset. By monotonicity of υ it suffices to show

$$(\sqcup\mathcal{Y})\upsilon Z \sqsubseteq_{EM} \sqcup\{Y\upsilon Z \mid Y\in\mathcal{Y}\}$$

which will be abbreviated to LHS $\sqsubseteq_{EM}$ RHS. When y is an element of Z it is immediate that y is an element of RHS because $y\in Y\upsilon Z$ for each $Y\in\mathcal{Y}$. When $y\in\sqcup\mathcal{Y}$ we have $Y\in\mathcal{Y}$ such that $y\in Y$ and hence $y\in Y\upsilon Z$. Also if $Y'\in\mathcal{Y}$ then $y\in RC(Y')$ and therefore $y\in RC(Y'\upsilon Z)$. This shows that y is an element of RHS. Conversely let y' be an element of RHS. If $y\in RC(Z)$ it is immediate to get y in LHS such that $y\sqsubseteq y'$. Otherwise $y'\in RC(Y)$ for every $Y\in\mathcal{Y}$. Since $\sqcup\mathcal{Y}$ is an upper bound of $\mathcal{Y}$ this shows that $(\sqcup\mathcal{Y})\upsilon\{y'\}$ also is. By theorem 5.2:2 $\sqcup\mathcal{Y}\sqsubseteq_{EM}(\sqcup\mathcal{Y})\upsilon\{y'\}$ and this gives y in LHS such that $y\sqsubseteq y'$. ///

When studying the relational powerdomain it was shown how to extend a strict and continuous function $f:D\to B$ to a completely additive function $f^\dagger:\mathcal{P}(D)\to B$. (This was used to define $\mathcal{P}$ as a functor and later to define the collecting semantics.) In the present setting $\mathcal{E}(D)$ and B may be viewed as having two partial orders. It will be useful to make use of the additional partial order in stating the properties of $f^\dagger$.

Let $B = (B,\sqsubseteq,\subseteq)$ and $C = (C,\sqsubseteq,\subseteq)$ be two augmented dcpo's and $g:B\to C$ a function. It is said to be <u>$\subseteq$-monotonic</u> iff $Y_1\subseteq Y_2$ implies $g(Y_1)\subseteq g(Y_2)$ for all elements Y_1 and Y_2 of B. If the augmentations of B and C are complete it is said to be <u>completely linear</u> if

$g(\bigcup Y) = \bigcup\{g(Y) \mid Y \in y\}$ holds for every nonempty subset y of B. We shall use $\subseteq$ and $\mathcal{L}$ to name these properties of functions and write $g:B \to_{\subseteq} C$ and $g:B \to_{\mathcal{L}} C$ respectively. The property of being $\subseteq$-monotonic is admissible if the augmentaion of C is admissible. Complete linearity needs not be an admissible property even if the augmentation of C is admissible. To see this define $g_i: \mathcal{E}(N_\perp) \to \mathcal{E}(N_\perp)$ by

$$g_i(Y) = \{\perp \mid \perp \in Y \vee \exists n \in Y: n \geq i\} \cup \{0 \mid \exists n \in Y: n < i\}$$

and $y = \{g_i \mid i \geq 1\}$. Then y is a directed set of completely linear functions but

$$\bigsqcup y = \lambda Y.\{\perp \mid \perp \in Y \text{ or } Y \text{ infinite}\} \cup \{0 \mid \exists n \in Y: n \neq \perp\}$$

is not completely linear.

For the extension of functions define the singleton function $\{\}: D \to \mathcal{E}(D)$ by $\{\}(d) = \{d\}$. It is clearly strict and monotonic.

Theorem 5.2:5. Let D be a benign fdcpo and B a completely augmented dcpo. For each monotonic $f:D \to B$ there exists precisely one monotonic and completely linear $f^\dagger: \mathcal{E}(D) \to B$ such that $f^\dagger \cdot \{\} = f$. It is given by $f^\dagger(Y) = \bigcup\{f(y) \mid y \in Y\}$ and $\lambda f.f^\dagger$ is a monotonic and $\subseteq$-monotonic bijection from $D \to B$ to $\mathcal{E}(D) \to_{\mathcal{L}} B$ and its inverse $\lambda g.g^* \{\}$ is also monotonic and $\subseteq$-monotonic. ///

Proof Suppose that g is a monotonic and completely linear function such that $g^* \{\} = f$. Then

$$g(Y) =$$
$$g(\bigcup\{\{y\} \mid y \in Y\}) =$$
$$\bigcup\{g(\{y\}) \mid y \in Y\} =$$
$$\bigcup\{f(y) \mid y \in Y\}$$

shows that g must equal $f^\dagger$. To see that $f^\dagger$ is as claimed note that $f^\dagger\cdot\{\} = f$ is immediate. For monotonicity of $f^\dagger$ let $Y_1 \sqsubseteq_{EM} Y_2$. By monotonicity of f

$$\{f(y) \mid y \in Y_1\} \sqsubseteq_{EM} \{f(y) \mid y \in Y_2\}$$

and by monotonicity of $\cup$ it follows that $f^\dagger(Y_1) \sqsubseteq f^\dagger(Y_2)$. For complete linearity of $f^\dagger$ calculate

$$f^\dagger(\cup y) = \cdots$$

(by $f^\dagger$ monotonic and $\cup y \sqsubseteq_{EM} \cup y \sqsubseteq_{EM} \cup y$)

$$f^\dagger(\cup y) =$$

$$\cup\{f(y) \mid y \in \cup y\} =$$

(by $\cup$ a least upper bound operator)

$$\cup\{\cup\{f(y) \mid y \in Y\} \mid Y \in y\} =$$

$$\cup\{f^\dagger(Y) \mid Y \in y\}$$

Clearly $\lambda f.f^\dagger$ is a bijection with inverse $\lambda g.g\cdot\{\}$. The inverse is obviously monotonic and $\sqsubseteq$-monotonic. To see that $\lambda f.f^\dagger$ is $\sqsubseteq$-monotonic let $f_1 \sqsubseteq f_2$, i.e. $\forall d: f_1(d) \sqsubseteq f_2(d)$. Then $f_1^\dagger(Y) = \cup\{f_1(d) \mid d \in Y\} \sqsubseteq \cup\{f_2(d) \mid d \in Y\} = f_2^\dagger(Y)$. To see that $\lambda f.f^\dagger$ is monotonic let $f_1 \sqsubseteq f_2$. Then $f_1^\dagger(Y) = \cup\{f_1(d) \mid d \in Y\} \sqsubseteq \cup\{f_2(d) \mid d \in Y\} = f_2^\dagger(Y)$ because $\cup$ is monotonic. ///

This theorem makes it possible to extend $\mathcal{P}$ to a covariant functor from the category BFDCPO of benign fdcpo's and monotonic functions to AUG of augmented dcpo's and monotonic and $\sqsubseteq$-monotonic functions. The effect upon objects has been defined and for a morphism $f:D \to E$ define $\mathcal{P}(f):\mathcal{P}(D) \to \mathcal{P}(E)$ by $\mathcal{P}(f) = (\{\}\cdot f)^\dagger = \lambda Y.\cup\{\{f(y)\} \mid y \in Y\}$. Clearly $\mathcal{P}(f)$ is in AUG and $\mathcal{P}(id) = id$. To see that $\mathcal{P}(f_1\cdot f_2) = \mathcal{P}(f_1)\cdot\mathcal{P}(f_2)$ it suffices by complete linearity to show that

$$\mathcal{G}(f_1 \cdot f_2) \cdot \{\} = \mathcal{G}(f_1) \cdot \mathcal{G}(f_2) \cdot \{\}$$

and this is straightforward.

For $g: \mathcal{G}(D) \to B$ a morphism of $\underline{\underline{\underline{AUG}}}$ define $LIN(g): \mathcal{G}(D) \to B$ by $LIN(g) = (g \cdot \{\})^\dagger$. Clearly this gives a completely linear morphism of $\underline{\underline{\underline{AUG}}}$ such that $LIN(g) \cdot \{\} = g \cdot \{\}$. As a function LIN is monotonic and $\subseteq$-monotonic by the theorem. It further follows from the theorem that $\mathcal{G}(D) \to_{\mathcal{L}} B$ is a dcpo with

$$\sqcup y =$$
$$(\sqcup \{g \cdot \{\} \mid g \in y\})^\dagger =$$
$$(\lambda d. \sqcup \{g(\{d\}) \mid g \in y\})^\dagger =$$
$$LIN(\lambda Y. \sqcup \{g(Y) \mid g \in y\})$$

The use of LIN cannot be avoided in general because complete linearity needs not be an admissible property upon $\mathcal{G}(D) \to B$ (and $\lambda Y. \sqcup \{g(Y) \mid g \in y\}$ is the least upper bound in $\mathcal{G}(D) \to B$).

It is because of the above theorem that no continuity assumptions are made. Clearly a function $f: D \to B$ from a benign fdcpo to an augmented dcpo is directed continuous because it is monotonic and D is of finite height. It is not the case, however, that $f^\dagger$ must be directed continuous. To see this define $f: N_\perp \times \{\tau\}_\perp \to \mathcal{G}(\{\tau\}_\perp)$ by $f(x,y) = \{y\}$. Further define the directed subset y of $\mathcal{G}(N_\perp \times \{\tau\}_\perp)$ by

$$y = \{ \{(1,\tau), \ldots, (j,\tau), (j+1,\perp), \ldots\} \mid j \geq 1 \}$$

Then $f^\dagger(Y) = \{\perp,\tau\}$ for $Y \in y$ so $\sqcup \{f^\dagger(Y) \mid Y \in y\}$ equals $\{\perp,\tau\}$. Since

$$\sqcup y = \{(1,\tau), \ldots\}$$

this differs from $f^\dagger(\sqcup y) = \{\tau\}$.

Remark For data flow analysis purposes it is natural to include the empty set as an element of the powerdomain. Then $\sqsubseteq_{EM}$ needs to be extended to a partial order $\sqsubseteq$ such that the previous development can still be carried through. It is not clear how to do so. For example if $\sqsubseteq$ is $\sqsubseteq_{EM}$ augmented with $\{\bot\}\sqsubseteq\emptyset$ and $\emptyset\sqsubseteq\emptyset$ it seems necessary to assume strictness in theorem 5.2:5 and this excludes modelling call-by-name as done in the next section. ///

...

Remark Clearly $\mathcal{P}(N_\bot)$ differs from the Smyth, Plotkin and relational powerdomains described in section 3.1. The definition of $\mathcal{P}(D)$ extends /ApP182/ where it was assumed that D was flat and countable. It is more difficult to compare $\mathcal{P}(D)$ with the powerdomains of /Plo82/ as these are not described as (a representation of) a collection of sets. (It is conjectured that this cannot be done, i.e. that contrary to $\mathcal{P}(D)$ there may be an element Y such that

$$\bigcup\{\{y\}\mid\{y\}\subseteq Y\}$$

does not equal Y.) However, a certain difference between the power-domains is shown below.

In /Plo82/ the powerdomain $\mathcal{P}(D)$ and singleton function $\{\}:D\to\mathcal{P}(D)$ defined there is shown to satisfy an analogue of theorem 5.2:5. This analogue is not fulfilled by $\mathcal{P}(D)$ and $\{\}:D\to\mathcal{P}(D)$. The crucial observation is that the extension $f^\dagger$ of f in /Plo82/ is always continuous (preserves least upper bounds of chains indexed by ω) but it has just been shown that this is not necessarily the case for the $f^\dagger$ guaranteed by 5.2:5. (We omit the detailed argument.) Similar considerations apply to the other powerdomain defined in /Plo82/. ///

5.3 STANDARD AND COLLECTING SEMANTICS

In this section we consider the applicative language of recursion equation schemes that was already defined in section 2.4. Its syntax is given by

$$\begin{aligned} \text{exp} ::= {} & x_i && (1 \le i \le k) \\ | \; & F_i(\text{exp}_1, \ldots, \text{exp}_k) && (1 \le i \le n) \\ & \ldots \\ | \; & A_i(\text{exp}_1, \ldots, \text{exp}_k) && (1 \le i) \end{aligned}$$

$$\begin{aligned} \text{pro} ::= {} & \underline{\text{let}}\ F_1(x_1, \ldots, x_k) = \text{exp}_1 \ \& \ \ldots \\ & \& \ F_n(x_1, \ldots, x_k) = \text{exp}_n \ \underline{\text{in}}\ \text{exp} \end{aligned}$$

and the parameter mechanism is intended to be call-by-name.

We begin with defining the semantics of this language. This is done relative to an interpretation as in the previous chapters but the notion of an interpretation will be simpler than before. Then the standard and collecting interpretations are defined. The standard semantics agrees with the one given in chapter 2 but the collecting semantics differs from the one of chapter 3 due to the richness of the present powerdomain. The latter is a straightforward consequence of a theorem (a simple analogue of 3.3:14) that shows the relation between the standard and collecting semantics.

An <u>interpretation</u> I consists of specifications of the components L_k, Q, L_1, take_i, tuple, a_i and $\square$. Actually take_i and a_i are families $(\text{take}_i)_{i=1,\ldots,k}$ and $(a_i)_i$ but we shall use the more informal notation. The conditions that they must satisfy are given below together with explanations of the analogous components in the previous chapters:

- L_k and L_1 are dcpo's; they correspond to the bottom-level constant domains and the intention is that $L_k = L_1 \underline{\times} \ldots \underline{\times} L_1$.
- Q is a predicate upon $L_k \to L_1$ such that $L_k \to_Q L_1$ is a dcpo; it corresponds to the bottom-level category.
- $take_i : L_k \to_Q L_1$ is much as before.
- $tuple : (L_k \to_Q L_1)^k \to L_k \to_Q L_k$ is much as before.
- $a_i : L_k \to_Q L_1$ corresponds to the constants of contravariantly pure type.
- $\square : (L_k \to_Q L_1) \times (L_k \to_Q L_k) \to L_k \to_Q L_1$ corresponds to composition in the bottom-level.

As before we shall write $\underline{I}(L_k)$ etc. when it is not evident which interpretation is considered. The dcpo L_k has been explicitly defined so there is no analogue of the bottom-level domain constructors and therefore it will not be necessary to study a tensor product. It is straightforward to generalise this to allow dcpo's L_{gt} for a class of types gt but this will not be necessary to illustrate the strong approach. It should be clear that $\ldots\underline{\times}\ldots$ corresponds to the use of top-level domain constructors.

The semantic functions

$$\underline{I}[\![exp]\!] : (L_k \to_Q L_1)^n \to (L_k \to_Q L_1)$$

$$\underline{I}[\![pro]\!] : L_k \to_Q L_1$$

is defined (relative to $\underline{I}$) much as before. The equations are:

$$\underline{I}[\![x_i]\!]env = take_i$$

$$\underline{I}[\![F_i(exp_1,\ldots,exp_k)]\!]env =$$

$$(env\downarrow i) \square tuple(\underline{I}[\![exp_1]\!]env, \ldots, \underline{I}[\![exp_k]\!]env)$$

$$\underline{I}[\![A_i(exp_1,\ldots,exp_k)]\!]env =$$

$$a_i \square tuple(\underline{I}[\![exp_1]\!]env, \ldots, \underline{I}[\![exp_k]\!]env)$$

$$\underline{I}[\![\underline{\text{let}} \ldots \underline{\text{in}}\ \text{exp}]\!] = \underline{I}[\![\text{exp}]\!](\text{LFP}(\text{abbr}_1))$$

$$\underline{\text{where}}\ \text{abbr}_1 = \lambda\text{env}.\ (\underline{I}[\![\text{exp}_1]\!]\text{env}\ ,\ \ldots,\ \underline{I}[\![\text{exp}_k]\!]\text{env})$$

The main difference from before is that continuity is not assumed but this causes no problems.

Fact 5.3:1. If $\underline{I}$ is an interpretation then $\underline{I}[\![\text{pro}]\!]$ and $\underline{I}[\![\text{exp}]\!]$ are defined with functionalities as stated. ///

...

The proof is straightforward by structural induction.

To define the standard interpretation $\underline{S}$ let S be some given set. Then put $L_1 = S_\perp$ and $L_k = S_\perp{}^k$ and let $a_i : S_\perp{}^k \to S_\perp$ be unspecified functions. Further let

$$\text{take}_i = \lambda(v_1,\ldots,v_k).\ v_i$$

$$\text{tuple}(f_1,\ldots,f_k) = \lambda(v_1,\ldots,v_k).$$

$$(f_1(v_1,\ldots,v_k),\ \ldots,\ f_k(v_1,\ldots,v_k))$$

$$f_1 \square f_2 = \lambda(v_1,\ldots,v_k).\ f_1(f_2(v_1,\ldots,v_k))$$

Finally, let Q be the constantly true predicate on $L_k \to L_1$. It is straightforward to verify that this gives an interpretation. One could as well have chosen Q to specify continuity because $S_\perp{}^k$ is of finite height and therefore every function in $L_k \to L_1$ is already continuous. Also tuple and $\square$ are directed continuous in their arguments so it is straightforward to prove by structural induction that

$$\underline{I}[\![\text{exp}]\!] : (L_k \to L_1)^n \to_C L_k \to L_1$$

i.e. that $\underline{I}[\![\text{exp}]\!](\text{env})$ depends continuously on env. This implies that, using abbr_1 above,

$$\text{LFP}(\text{abbr}_1) = \bigsqcup_{m=0}^{\infty} \text{abbr}_1{}^m(\perp)$$

and it is then immediate that this standard semantics agrees with the one of chapter 2.

The aim with the collecting interpretation $\underline{C}$ is to achieve that $\underline{C}[\![pro]\!] = \mathcal{E}(\underline{S}[\![pro]\!])$. To obtain this we define $\underline{C}$ by:

$$L_k = \mathcal{E}(\underline{S}(L_k)) = \mathcal{E}(S_\perp^k)$$

$$L_1 = \mathcal{E}(\underline{S}(L_1)) = \mathcal{E}(S_\perp)$$

$$Q(g) \equiv g \text{ is } \subseteq\text{-monotonic}$$

$$take_i = \mathcal{E}(\underline{S}(take_i)) = \lambda Y.\ \{v_i \mid (v_1,\ldots,v_k) \in Y\}$$

$$tuple(g_1,\ldots,g_k) = LIN(\lambda Y.g_1(Y) \times \ldots \times g_k(Y))$$

$$a_i = \mathcal{E}(\underline{S}(a_i))$$

$$g_1 \square g_2 = LIN(g_1 \cdot g_2)$$

It follows from theorem 5.2:2 that $\mathcal{E}(S_\perp)$ and $\mathcal{E}(S_\perp^k)$ both are dcpo's. Also $\subseteq$-monotonicity is an admissible predicate on $\mathcal{E}(S_\perp^k) \to \mathcal{E}(S_\perp)$ because the augmentation of $\mathcal{E}(S_\perp)$ is admissible. It is then straightforward to verify that $\underline{C}$ is an interpretation.

Apart from the definition of $\square$ the definitions are much as might be expected from chapter 3. No tensor products are considered but the main purpose of the tensor product was to obtain $\mathcal{E}(S_\perp^k)$ from k copies of $\mathcal{E}(S_\perp)$ so direct use of $\mathcal{E}(S_\perp^k)$ is satisfactory for the collecting semantics. Since $\subseteq$ represents "safe approximation" it is natural that Q expresses some property with respect to this partial order. Because $\underline{C}[\![pro]\!]$ is to be $\mathcal{E}(\underline{S}[\![pro]\!])$, which is completely linear, it might be natural to let Q specify complete linearity. This is analogous to complete additivity in the previous chapters but it was decided against doing so because complete additivity does not hold in general for abstract interpretation. Similar considerations apply here and $\subseteq$-monotonicity is used because it can be enforced in general.

The explanation of the use of LIN in the definition of □ will be postponed because the following result is not affected by the removal of LIN. The result expresses a weaker connection between $\underline{S}$ and $\underline{C}$ than the desired $\underline{C}[\![pro]\!] = \mathcal{E}(\underline{S}[\![pro]\!])$, namely that $\underline{C}[\![pro]\!]$ is pointwise correct with respect to $\underline{S}[\![pro]\!]$:

Lemma 5.3:2. $\underline{C}[\![pro]\!]\cdot\{\} = \{\}\cdot\underline{S}[\![pro]\!]$ ///

Proof Let the predicates R and Rn be defined by

$$R(f,g) \equiv g\cdot\{\} = \{\}\cdot f$$

$$Rn(senv,cenv) \equiv \forall j:\ R(senv\downarrow j, cenv\downarrow j)$$

It is a straightforward structural induction to prove that

$$Rn(ienv,cenv) \Rightarrow R(\ \underline{I}[\![exp]\!](ienv)\ ,\ \underline{C}[\![exp]\!](cenv)\ )$$

(whether or not LIN is used in the definition of $\underline{C}(\square)$). To show the result it therefore suffices to show

$$Rn(LFP(F),LFP(G))$$

for the obvious functionals F and G. This result follows from the induction principle of section 5.1 if Rn is an admissible predicate. Clearly Rn is admissible iff R is. It is immediate that $R(\perp,\perp)$ holds so let y be a directed subset of $(S_\perp^k \to S_\perp) \times (\mathcal{E}(S_\perp^k) \to_s \mathcal{E}(S_\perp))$ such that R(f,g) holds for each $(f,g)\in y$. To see $R(\sqcup y)$, i.e. $R((\sqcup y)\downarrow 1, (\sqcup y)\downarrow 2)$, we calculate:

$$(\sqcup y)\downarrow 2\cdot\{\} =$$

$$(\sqcup\{g \mid \exists f:\ (f,g)\in y\})\cdot\{\} =$$

(pointwise least upper bounds in $\mathcal{E}(S_\perp^k) \to_{\subseteq} \mathcal{E}(S_\perp)$)

$$\sqcup\{g\cdot\{\} \mid \exists f:\ (f,g)\in y\} =$$

$$\sqcup\{\ \{\}\cdot f \mid \exists g:\ (f,g)\in y\} =$$

({} is directed continuous)

$$\{\}\cdot\sqcup\{f \mid \exists g:\ (f,g)\in y\} =$$
$$\{\}\cdot(\sqcup y)\downarrow 1 \qquad ///$$

The use of LIN in the definition of $\underline{C}(\square)$ is one way of achieving that $\underline{C}[\![pro]\!]$ is completely linear and the desired result then follows from the previous lemma:

...

Theorem 5.3:3. $\underline{C}[\![pro]\!] = \mathcal{E}(\underline{S}[\![pro]\!])$ ///

Proof It is immediate that the placement of $\square$ ensures that $\underline{C}[\![exp]\!](cenv)$ always is completely linear. Therefore $\underline{C}[\![pro]\!]$ is completely linear as well and equals

$$(\underline{C}[\![pro]\!]\cdot\downarrow)^{\dagger}$$

Using the previous lemma this equals $\mathcal{E}(\underline{S}[\![pro]\!])$. ///

There are other ways that can be used to obtain that $\underline{C}[\![pro]\!]$ is completely linear. The present choice was inspired by /Hen82/ that considers

$$g_1 \square g_2 = LIN(g_1)\cdot g_2$$

(for some notion of LIN). In the present setting g_2 will always be tuple(...), hence completely linear, so $LIN(g_1)\cdot g_2$ agrees with $LIN(g_1\cdot g_2)$. Another possibility is to omit LIN from the definition of $\square$ and let Q be complete linearity (as was decided against above). A third and closely related possibility is to omit LIN from the definition of $\square$ and replace LFP(G) by LINn(LFP(G)) where $LINn(g_1,\ldots,g_n) = (LIN(g_1),\ldots,LIN(g_n))$. It can be shown that all possibilities give the same results and that for the present choice

$$LFP(G) = LINn(\bigsqcup_{m=0} G^m(\bot)) = G(\bigsqcup_{m=0} G^m(\bot))$$

We omit the proof: it relies on the continuity of $\underline{S}[\![\exp]\!]$, the proof of the induction principle of section 5.1 and the characterisation in section 5.2 of least upper bounds in $\mathcal{P}(S_\perp^k) \to_{\mathcal{L}} \mathcal{P}(S_\perp)$.

5.4 ABSTRACT INTERPRETATION

As in the previous chapters data flow analyses are specified by certain interpretations (called approximating interpretations). In this section it is studied how to prove the correctness of an approximating semantics with respect to the collecting semantics or another approximating semantics. We also study how to induce an approximating interpretation from the collecting interpretation or another approximating interpretation. The framework differs from that of the previous chapters because $\mathcal{P}(S_\perp^k)$ and $\mathcal{P}(S_\perp)$ are naturally equipped with two partial orders and we shall argue that this should be the case for all approximation spaces.

Approximating interpretation

So let $\underline{I} = (L_k, Q, L_1, take_i, tuple, a_i, \square)$ be an interpretation that expresses some data flow analysis. The connection between L_k and $\mathcal{P}(S_\perp^k)$ and between L_1 and $\mathcal{P}(S_\perp)$ will be given by concretization functions:

$$con_k : L_k \to \mathcal{P}(S_\perp^k)$$

$$con_1 : L_1 \to \mathcal{P}(S_\perp)$$

The desired relation between $\underline{I}$ and the collecting interpretation $\underline{C}$ is that $\underline{I}[\![pro]\!]$ should be a safe description of $\underline{C}[\![pro]\!]$. We shall formalise this by

$$\underline{C}[\![pro]\!] \cdot con_k \sqsubseteq con_1 \cdot \underline{I}[\![pro]\!]$$

rather than

$$\underline{C}[\![pro]\!]\cdot con_k \sqsubseteq_{EM} con_1 \cdot \underline{I}[\![pro]\!]$$

as was used in the previous chapters. (Here $\sqsubseteq$ and $\sqsubseteq_{EM}$ are extended pointwise to functions.) This is because it is $\sqsubseteq$ and not $\sqsubseteq_{EM}$ that expresses "safe approximation": it is evident that $\sqsubseteq$ expresses that every result that is produced by the collecting semantics is included among those produced by the abstracting semantics. This is not the case with $\sqsubseteq_{EM}$: it is problematic that $\{1,2\} \sqsubseteq_{EM} \{2\}$ holds because then non-termination may be ignored, and it is problematic that $\{2\} \sqsubseteq_{EM} \{2,3\}$ fails because this makes it almost impossible to approximate.

Not all interpretations specify data flow analyses and we now redefine the notion of approximating interpretation to include those that do. The collecting interpretation is to be one and we have seen that both $\sqsubseteq_{EM}$ and $\sqsubseteq$ were relevant for $\mathcal{S}(S_\perp^k)$ and $\mathcal{S}(S_\perp)$. If $\underline{I}$ is to be an approximating interpretation it is natural that L_k and L_1 also have two partial orders. One is written $\sqsubseteq$ and is used to guarantee the existence of least fixed points in the same way that $\sqsubseteq_{EM}$ has been used. The other is written $\subseteq$ and expresses safe approximation in much the same way that subset inclusion does. The intuitive difference between $\sqsubseteq$ and $\subseteq$ was already mentioned in /Nie81a/ and /Nie82/.

Since L_k and L_1 are equipped with the partial order $\subseteq$ it is natural to let Q express some property with respect to $\subseteq$. We shall use $\subseteq$-monotonicity because it seems obvious that performing a computation upon a safe description of some argument must give a result that safely describes the result of computing upon the original

argument. Stronger preservation properties such as complete linearity may fail in general and for much the same reasons that additivity could not be assumed in general in chapter 4. The partial order can be extended pointwise so as to relate functions in $L_k \to L_1$. It is natural to assume that tuple and $\sqcup$ are $\sqsubseteq$-monotonic in each argument. When $\sqsubseteq$ is extended componentwise to cartesian products this just means that tuple and $\sqcup$ are $\sqsubseteq$-monotonic.

...

This discussion motivates:

Definition An approximating interpretation is a specification of L_k, Q, L_1, $take_i$, tuple, a_i, $\sqcup$ such that:

- L_k and L_1 are augmented dcpo's with the augmentation of L_1 admissible,
- Q specifies $\sqsubseteq$-monotonicity,
- $take_i : L_k \to_{\sqsubseteq} L_1$ and $a_i : L_k \to_{\sqsubseteq} L_1$,
- tuple: $(L_k \to_{\sqsubseteq} L_1)^k \to_{\sqsubseteq} (L_k \to_{\sqsubseteq} L_k)$ and $\sqcup : (L_k \to_{\sqsubseteq} L_1) \times (L_k \to_{\sqsubseteq} L_k) \to_{\sqsubseteq} (L_k \to_{\sqsubseteq} L_1)$

///

The predicate Q is admissible upon $L_k \to L_1$ because the augmentation of L_1 is admissible. It is therefore straightforward to prove that an approximating interpretation is an interpretation and that least upper bounds of directed subsets of $L_k \to_{\sqsubseteq} L_1$ are pointwise (as in $L_k \to L_1$). If <u>if</u>...<u>then</u>....<u>else</u>... had been an explicit construct in the language it would be convenient to assume that the augmentation was complete. Finally, note that the collecting interpretation is an approximating interpretation.

Relating two interpretations

Next we show how to relate approximating interpretations $\underline{I}$ (as above) and $\underline{I}' = (L_k', Q', L_1', take_i', tuple', a_i', \sqcap')$ in such a way that $\underline{I}'[\![pro]\!]$ is a safe approximation to $\underline{I}[\![pro]\!]$. To express the relation we need concretization functions

$$con_k : L_k' \to L_k$$

$$con_1 : L_1' \to L_1$$

and we shall refer to them as $con = (con_i)_{i=1,k}$. The desired relation between $\underline{I}[\![pro]\!]$ and $\underline{I}'[\![pro]\!]$ is $\leq_{con}$ defined on $(L_k \xrightarrow[\subseteq]{} L_1) \times (L_k' \xrightarrow[\subseteq]{} L_1')$ by

$$g \leq_{con} g' \quad \text{iff} \quad g \cdot con_k \subseteq con_1 \cdot g'$$

We shall also need $\leq_{con}$ to be defined on $(L_k \xrightarrow[\subseteq]{} L_k) \times (L_k' \xrightarrow[\subseteq]{} L_k')$ by a similar formula. The relation may be extended to approximating interpretations by defining $\underline{I} \leq_{con} \underline{I}'$ to hold whenever

- $take_i \leq_{con} take_i'$ and $a_i \leq_{con} a_i'$ (for all i)
- $g_i \leq_{con} g_i'$ (for all i) implies

  $tuple(g_1,\ldots,g_k) \leq_{con} tuple'(g_1',\ldots,g_k')$ and

  $g_i \leq_{con} g_i'$ (for i=1,2) implies $g_1 \sqcap g_2 \leq_{con} g_1' \sqcap' g_2'$

This definition closely follows the pattern of chapter 4.

To prove that $\underline{I}[\![pro]\!] \leq_{con} \underline{I}'[\![pro]\!]$ we shall need more assumptions than merely $\underline{I} \leq_{con} \underline{I}'$. These express conditions about how con_1 preserves aspects of the partial order $\subseteq$ with respect to the partial order $\subseteq$. We say con_1 is pseudo-strict iff $\perp \subseteq con_1(\perp)$. If $L_1 = \mathcal{P}(S_\perp)$ it is evident that this means that $\perp$ of L_1' must represent the possibility of a non-terminating outcome. The function con_1 is pseudo-continuous iff

$$\sqcup\{con_1(Y) \mid Y \in \mathcal{Y}\} \sqsubseteq con_1(\sqcup\mathcal{Y})$$

holds for every directed subset $\mathcal{Y}$ of L_1'. (This is essentially the dual notion of the quasi-continuity considered in /Plo82/.) It is immediate that strictness and directed continuity imply pseudo-strictness and pseudo-continuity, respectively.

Theorem 5.5:1. If $\underline{I} \preceq_{con} \underline{I}'$ and con_1 is pseudo-strict and pseudo-continuous then $\underline{I}[\![pro]\!] \preceq_{con} \underline{I}'[\![pro]\!]$ holds for all programs pro. ///

Proof First extend the relation $\preceq_{con}$ to tuples by defining $env \preceq env'$ to mean that $env\downarrow i \preceq_{con} env'\downarrow i$ holds for all i. It is then a straightforward structural induction to show that

$$env \preceq env' \text{ implies } \underline{I}[\![exp]\!](env) \preceq_{con} \underline{I}'[\![exp]\!](env')$$

To show the result for pro = let ... in exp_0 it therefore suffices to show that $LFP(G) \preceq LFP(G')$ where

$$G(env) = (\underline{I}[\![exp_1]\!](env), \ldots, \underline{I}[\![exp_n]\!](env))$$

and G' is defined similarly. Since $env \preceq env'$ implies that $G(env) \preceq G'(env')$ it suffices to show that $\preceq$ is admissible as the result then follows from the induction principle of section 5.1.

The relation $\preceq$ is admissible iff $\preceq_{con}$ is upon $(L_k \xrightarrow[\preceq]{} L_1) \times (L_k' \xrightarrow[\preceq]{} L_1')$. It is immediate that $\perp \preceq_{con} \perp$ because con_1 is pseudo-strict. Next let $\mathcal{Y}$ be a directed set such that $g \preceq_{con} g'$ holds for all $(g,g') \in \mathcal{Y}$. For $(\sqcup\mathcal{Y})\downarrow 1 \preceq_{con} (\sqcup\mathcal{Y})\downarrow 2$ calculate as follows:

$$\forall (g,g') \in \mathcal{Y}: g \preceq_{con} g' \Rightarrow$$

$$\forall (g,g') \in \mathcal{Y}: g \cdot con_k \preceq con_1 \cdot g' \Rightarrow$$

($\preceq$ is admissible on L_1)

$$\sqcup\{g \cdot con_k \mid (g,g') \in \mathcal{Y}\} \preceq \sqcup\{con_1 \cdot g' \mid (g,g') \in \mathcal{Y}\} \Rightarrow$$

(pointwise least upper bounds)

$$(\sqcup y)\downarrow 1 \cdot con_k \subseteq \sqcup\{con_1 \cdot g' \mid (g,g') \in y\} \Rightarrow$$

(con_1 is pseudo-continuous)

$$(\sqcup y)\downarrow 1 \cdot con_k \subseteq con_1 \cdot (\sqcup y)\downarrow 2 \Rightarrow$$

$$(\sqcup y)\downarrow 1 \leq_{con} (\sqcup y)\downarrow 2$$ ///

Inducing an interpretation

Also the notion of inducing an interpretation may be defined in the strong approach. To do so requires abstraction functions

$$abs_k : L_k \rightarrow L_k'$$

$$abs_1 : L_1 \rightarrow L_1'$$

in addition to the concretization functions con_k and con_1. In the previous chapter "adjointness" was used to express a desirable connection between abstraction and concretization functions. Here "adjointness" should be defined with respect to $\subseteq$ because it is this partial order that expresses "safe approximation". It is reasonable to assume that abstraction and concretization functions are $\subseteq$-monotonic because this means that they preserve "safe approximation". We then redefine:

Definition A pair $(abs:L \rightarrow L', con:L' \rightarrow L)$ of functions between augmented dcpo's is a pair of adjoined functions iff abs and con are $\subseteq$-monotonic and monotonic and

$$abs \cdot con \subseteq \lambda 1'.1' \quad \text{and} \quad con \cdot abs \supseteq \lambda 1.1$$

Function abs is said to be the lower adjoint of con and con is the upper adjoint of abs. ///

As before, lower and upper adjoints are unique if they exist. It will

be used below that they must be monotonic.

Let $\underline{I}$ be an approximating interpretation with components as above and let L'_k and L'_1 be augmented dcpo's such that the augmentation of L'_1 is admissible. Further let $(abs_k : L_k \to L'_k, con_k)$ and $(abs_1 : L_1 \to L'_1, con_1)$ be pairs of adjoined functions. The induced interpretation

$$\underline{I}' = induce(\underline{I}, (abs_i, con_i)_i, (L'_i)_i)$$

has $\underline{I}'(L_k) = L'_k$, $\underline{I}'(L_1) = L'_1$ and $\underline{I}'(Q)$ to be $\subseteq$-monotonicity. The remaining components are given by:

$$take'_i = abs_1 \cdot take_i \cdot con_k$$

$$tuple'(g_1,\ldots,g_k) = abs_k \cdot tuple(con_1 \cdot g_1 \cdot abs_k, \ldots) \cdot con_k$$

$$a'_i = abs_1 \cdot a_i \cdot con_k$$

$$g_1 \square' g_2 = abs_1 \cdot ((con_1 \cdot g_1 \cdot abs_k) \square (con_k \cdot g_2 \cdot abs_k)) \cdot con_k$$

It is straightforward to verify that this gives an approximating interpretation. The following lemma shows that $\underline{I}'$ is correct and as precise as possible:

Lemma 5.4:2. Let $\underline{I}$, L'_i, abs_i and con_i be as above. Then

(1) $\underline{I} \lesssim_{con} induce(\underline{I}, (abs_i, con_i)_i, (L'_i)_i)$

(2) $\underline{I} \lesssim_{con} \underline{I}''$ iff $induce(\underline{I}, (abs_i, con_i)_i, (L'_i)_i) \lesssim_{id} \underline{I}''$

where id is the family of identity functions. ///

Proof For (1) let $\underline{I}'$ be defined as above. It is immediate that $take_i \lesssim_{con} take'_i$ and $a_i \lesssim_{con} a'_i$. Next let $g_i \lesssim_{con} g'_i$ so that

$$g_i \subseteq abs_1 \cdot g'_i \cdot con_k$$

By $\subseteq$-monotonicity of tuple

$$tuple(g_1,\ldots,g_k) \subseteq tuple(abs_1 \cdot g'_1 \cdot con_k, \ldots)$$

from which it follows that

$$\text{tuple}(g_1,\ldots,g_k) \subseteq \text{tuple}'(g_1',\ldots,g_k')$$

That $g_i \preceq_{con} g_i'$ implies $g_1 \square g_2 \preceq_{con} g_1' \square' g_2'$ follows similarly.

For (2) let $\underline{I}'' = (L_k', Q', L_1', \text{take}_i'', \text{tuple}'', a_i'', \square'')$ and consider "if". When $\underline{I}' \preceq_{id} \underline{I}''$ it is immediate that $a_i \preceq_{con} a_i''$ and $\text{take}_i \preceq_{con} \text{take}_i''$. Next let $g_i \preceq_{con} g_i''$ so that

$$\text{abs}_1 \cdot g_i \cdot \text{con}_k \subseteq g_i''$$

Then by $\underline{I}' \preceq_{id} \underline{I}''$

$$\text{tuple}'(\text{abs}_1 \cdot g_1 \cdot \text{con}_k, \ldots) \subseteq \text{tuple}''(g_1'', \ldots)$$

and using the definition of tuple' this gives

$$\text{tuple}(\text{con}_1 \cdot \text{abs}_1 \cdot g_1 \cdot \text{con}_k \cdot \text{abs}_k, \ldots) \cdot \text{con}_k \subseteq \text{con}_1 \cdot \text{tuple}''(g_1'', \ldots)$$

By $\subseteq$-monotonicity of tuple this gives

$$\text{tuple}(g_1,\ldots,g_k) \preceq_{con} \text{tuple}''(g_1'',\ldots,g_k'')$$

In a similar way it is shown that $g_i \preceq_{con} g_i''$ implies that $g_1 \square g_2 \preceq_{con} g_1'' \square'' g_2''$. It follows that $\underline{I} \preceq_{con} \underline{I}''$.

For "only if" in (2) assume that $\underline{I} \preceq_{con} \underline{I}''$. It is immediate that $a_i' \preceq_{id} a_i''$ and $\text{take}_i' \preceq_{id} \text{take}_i''$. Next let $g'_i \preceq_{id} g_i''$ so that

$$\text{con}_1 \cdot g_i' \cdot \text{abs}_k \preceq_{con} g_i''$$

This gives

$$\text{tuple}(\text{con}_1 \cdot g_1' \cdot \text{abs}_k, \ldots) \preceq_{con} \text{tuple}''(g_1'', \ldots)$$

and

$$\text{tuple}'(g_1',\ldots,g_k') \preceq_{id} \text{tuple}''(g_1'',\ldots,g_k'')$$

follows much as in the proof of "if". The result for $\square'$ and $\square''$ is

similar and $\underline{I}' \leq_{id} \underline{I}''$ follows. ///

If additionally con_1 is pseudo-strict and pseudo-continuous it follows by theorem 5.4:1 that $\underline{I}'[\![pro]\!]$ is correct with respect to $\underline{I}[\![pro]\!]$.

As in the previous chapter it is helpful to be able to construct data flow analyses in stages. It is straightforward to prove that if

$$\underline{I}' = induce(\underline{I}, (abs_i, con_i)_i, (L'_i)_i)$$

$$\underline{I}'' = induce(\underline{I}', (abs'_i, con'_i)_i, (L''_i)_i)$$

then

$$\underline{I}'' = induce(\underline{I}, (abs'_i, con'_i)_i \cdot (abs_i, con_i)_i, (L''_i)_i)$$

where $(abs'_i, con'_i)_i \cdot (abs_i, con_i)_i$ equals $(abs'_i \cdot abs_i, con_i \cdot con'_i)_i$. This shows that one obtains the same result whether or not the construction is "in stages".

Similar results hold for proving approximating interpretations correct. Here

$\underline{I} \leq_{con} \underline{I}'$ and $\underline{I} \leq_{con'} \underline{I}''$ and con and con' families of upper adjoints imply that $\underline{I} \leq_{con \cdot con'} \underline{I}''$

where $(con_i)_i \cdot (con'_i)_i = (con_i \cdot con'_i)_i$. The proof is analogous to the proof of "if" for the second statement of lemma 5.4:2. That theorem 5.4:1 will be applicable for $\underline{I}$ and $\underline{I}''$ if it is for $\underline{I}$ and $\underline{I}'$ and for $\underline{I}'$ and $\underline{I}''$, follows from:

Fact 5.4:3. If con_1 and con'_1 are pseudo-strict and pseudo-continuous upper adjoints then so is $con_1 \cdot con'_1$. ///

5.5 APPLICATIONS

The major application in this section is to show the correctness of two data flow analyses used in /Myc81/ for detecting situations where call-by-name can safely be replaced by call-by-value. First, however, we show the connection between the "strong" approach of this chapter and the "weak" approach of the previous chapters. This also gives a connection with the traditional theory of abstract interpretation (/CoCo77b/,...).

"Strong" versus "weak"

Let $\underline{C}$ be the collecting interpretation as defined in section 5.3. We define an approximating interpretation $\underline{W}$ that is like the collecting interpretation of chapter 3 (for the applicative language considered in section 2.4). It is obvious that $\underline{W}(L_1)$ should be $\wp(S_\perp)$. For this to make sense $S_\perp$ must be an algebraic cpo so we must assume that S is countable. Clearly $\wp(S_\perp)$ is a dcpo but it is not an augmented dcpo because $\wp(S_\perp) = (\wp(S_\perp), \subseteq)$ is only equipped with one partial order (which happens to be subset-inclusion). Since $\subseteq$ is used both to obtain least fixed points and to express safeness it should be turned into an augmented dcpo by setting

$$\wp(S_\perp) = (\wp(S_\perp), \subseteq, \subseteq)$$

and clearly the augmentation is admissible. According to chapter 3 $\underline{W}(L_k)$ should be $\wp(S_\perp) \otimes \ldots \otimes \wp(S_\perp)$ but since this is isomorphic to $\wp(S_\perp{}^k)$ we shall use the latter. It is made into an augmented dcpo as above. The predicate $\underline{W}(Q)$ is $\subseteq$-monotonicity and this reduces to monotonicity.

It is then straightforward to define the remaining components (building on the definition in section 3.3):

$$\underline{W}(\text{take}_i) = \lambda Y.\{y_i \mid (y_1,\ldots,y_k) \in Y\}$$

$$\underline{W}(\text{tuple})(g_1,\ldots,g_k) = \lambda Y.$$

$$\quad LC(\cup\{g_1(\{y\})\times\ldots\times g_k(\{y\}) \mid y \in Y\})$$

$$\underline{W}(a_i) = \lambda Y.\ LC(\{\underline{S}(a_i)(y_1,\ldots,y_k) \mid (y_1,\ldots,y_k) \in Y\})$$

$$\underline{W}(\square)(g_1,g_2) = g_1 \cdot g_2$$

It is straightforward to verify that $\underline{W}$ is an approximating interpretation. It should also be clear that it corresponds to the collecting interpretation of chapter 3. In particular it makes no harm that $\underline{W}(Q)$ does not specify continuity as in chapter 3 because continuity is an admissible property so the least fixed points will be the same.

To show that the present approach subsumes that of the previous chapters we first show that $\underline{W}$ can be proved correct with respect to $\underline{C}$. For this define the concretization functions

$$con_i : \wp(S_\perp^i) \to \mathcal{E}(S_\perp^i)$$

(where $S_\perp^1$ is $S_\perp$) by $con_i(Y) = Y$. The abstraction functions are defined by $abs_i(Y) = LC(Y)$. Then (abs_1,con_1) and (abs_k,con_k) are pairs of adjoined functions in the sense of section 5.4. For monotonicity note that $Y_1 \subseteq Y_2$ implies $Y_1 \sqsubseteq_{EM} Y_2$ when Y_1 contains $\perp$ and that $Y_1 \sqsubseteq_{EM} Y_2$ implies $LC(Y_1) \subseteq LC(Y_2)$ and hence $Y_1 \sqsubseteq Y_2$ when Y_1 and Y_2 are elements of $\wp(S_\perp^i)$. Further con_1 is pseudo-strict and pseudo-continuous. For pseudo-continuity it suffices by admissibility of $\subseteq$ upon $\mathcal{E}(S_\perp)$ to show that

$$con_1(Y) \subseteq con_1(\sqcup y)$$

whenever Y is an element of a directed set $\mathcal{Y}$ of $\mathcal{P}(S_\perp^i)$ and this is immediate by $\sqcup\mathcal{Y} = \cup\mathcal{Y}$. It is straightforward to show that $\underline{C} \preceq_{con} \underline{W}$. so theorem 5.4:1 shows that $\underline{C}[\![pro]\!] \preceq_{con} \underline{W}[\![pro]\!]$ holds for all programs pro.

To fully show that the present approach subsumes the previous approach it must be shown how pairs of adjoined functions and $\preceq_{con}$ carry over. So let $\underline{I}$ and $\underline{I}'$ be approximating interpretations in the sense of chapter 4. They can be turned into approximating interpretations $\underline{I}$ and $\underline{I}'$ in the sense of section 5.4 in the same way $\underline{W}$ was obtained. Next let $(abs_{gt}, con_{gt})_{gt}$ be a family of pairs of adjoined functions (in the sense of chapter 4) from $\underline{I}$ to $\underline{I}'$. They will still be pairs of adjoined functions in the sense of section 5.4 and the assumed strictness and continuity of con_{L_1} gives pseudo-strictness and pseudo-continuity. Further if $\underline{I} \preceq_{con} \underline{I}'$ in the sense of chapter 4 then this relation still holds in the sense of section 5.4.

Remark To compare this approach with the traditional theory of e.g. /CoCo77b/ it might be best to restrict k to 1 and augment the language with an explicit conditional. This is because the applicative language then closely corresponds to a flowchart. Alternatively $\mathcal{P}(S_\perp \times \ldots \times S_\perp)$ can be replaced by $\mathcal{P}(S_\perp * \ldots * S_\perp)$ as the latter is isomorphic to the powerset of S^k. Then one should use

$$con_k(Y) = Y \cup \bigcup_{i=1}^{k} S_\perp \times \ldots \times \{\perp\} \times \ldots \times S_\perp$$

$$abs_k(Y) = \{smash(y) \mid y \in Y\}$$

instead of the definitions above. ///

The data flow analyses from /Myc81/

We now turn towards the major application of this section. We shall formulate and prove correct two analyses used in /Myc81/. The first can be handled in the weak approach of the previous chapters whereas this seems impossible for the second. Both analyses were "proved correct" in /Myc81/ but the proof of the second fails (as will be pointed out) because it is based on the Plotkin powerdomain.

It is convenient to define the two analyses by inducing them from the following interpretation $\underline{I}$ that is essentially an independent attribute method formulation of the collecting interpretation.

$$L_k = \mathcal{P}(S_\perp)^k$$

$$L_1 = \mathcal{P}(S_\perp)$$

Q is constantly true

$$take_i = \lambda(Y_1,\ldots,Y_k).Y_i$$

$$tuple(g_1,\ldots,g_k) = \lambda(Y_1,\ldots,Y_k).\ (g_1(Y_1,\ldots,Y_k),\ldots,g_k(Y_1,\ldots,Y_k))$$

$$a_i = \lambda(Y_1,\ldots,Y_k).\ \{\underline{S}(a_i)(y_1,\ldots,y_k) \mid \forall i:\ y_i \in Y_i\}$$

$$g_1 \square g_2 = g_1 \dot{} g_2$$

It should not be surprising that LIN has been removed from tuple because $\underline{I}$ is an independent attribute method and it is convenient to remove it from $\square$ as well. It is straightforward to verify that this gives an approximating interpretation. The connection with the collecting interpretation is given by

$$con_k = \lambda(Y_1,\ldots,Y_k).\ Y_1 \times \ldots \times Y_k$$

$$con_1 = \lambda Y.Y$$

It is straightforward to prove that con_1 is pseudo-strict and pseudo-continuous and that $\underline{C} \lesssim_{con} \underline{I}$.

We shall formalise the first data flow analysis (# of /Myc81/) as an approximating interpretation $\underline{L}$. Let $\mathbb{O}$ be the set $\{0,1\}$. The elements of $\underline{L}(L_1)$ and $\underline{L}(L_k)$ are to be $\mathbb{O}$ and $\mathbb{O}^k$ respectively. The intention is that 0 represents guaranteed non-termination whereas 1 represents possible termination. So if

$$\underline{L}[\![pro]\!](0,1,\ldots,1) = 0$$

this means that the program cannot terminate unless its first argument represents a terminating computation. This again means that the first parameter (which is call-by-name) can safely be passed by value. Consult /Myc81/ for further information.

To define $\underline{L}$ as induced from $\underline{I}$ we must turn $\mathbb{O}$ and $\mathbb{O}^k$ into augmented dcpo's and define concretization and abstraction functions. On $\mathbb{O}$ there is the partial order $\leq$ (less than or equal to) and it may be extended componentwise to $\mathbb{O}^k$. Then $\underline{L}(L_1)$ and $\underline{L}(L_k)$ will be $(\mathbb{O},\leq,\leq)$ and $(\mathbb{O}^k,\leq,\leq)$ respectively and clearly they are admissibly augmented dcpo's. The concretization functions

$$con_i' : (\mathbb{O}^i,\leq,\leq) \to \underline{I}(L_i) \qquad i=1,k$$

(for $\mathbb{O}^1 = \mathbb{O}$) are defined by

$$con_1'(0) = \{\perp\}\ ,\ con_1'(1) = S_\perp$$

$$con_k'(v_1,\ldots,v_k) = (con_1'(v_1),\ldots,con_1'(v_k))$$

It is immediate that con_1' is pseudo-strict and pseudo-continuous.

Pairs of adjoined functions (abs_1',con_1') and (abs_k',con_k') may be obtained by defining $abs_1'(Y) = 0$ if $Y = \{\perp\}$ and $abs_1'(Y) = 1$ otherwise and $abs_k'(Y_1,\ldots,Y_k) = (abs_1'(Y_1),\ldots,abs_1'(Y_k))$. Then $\underline{L}$ is defined by

$$\underline{L} = induce(\underline{I},(abs_i',con_i')_i,((\mathbb{O}^i,\leq,\leq))_i)$$

Clearly $\underline{L}$ is correct with respect to $\underline{C}$, i.e. $\underline{C} \leq_{con\cdot con'} \underline{L}$. Further

the conditions of theorem 5.4:1 are fulfilled so that

$$\mathcal{E}(\underline{S}[\![pro]\!]) \cdot con_k \cdot con'_k \subseteq con_1 \cdot con'_1 \cdot \underline{L}[\![pro]\!]$$

holds. This implies that the informal description of $\underline{L}$ that was given above is correct.

Example Let pro be the following program:

$\underline{let}\ F_1(x_1,x_2,x_3,x_4,x_5) =$

$\quad \underline{if}\ x_1=1\ \underline{then}\ x_2+x_3\ \underline{else}\ F_1(x_1-1,x_3,x_2,x_5,x_4)$

$\underline{in}\ F_1(x_1,x_2,x_3,x_4,x_5)$

Formally this is not a program because all of $x_1=1$, x_2+x_3, x_1-1 and $\underline{if}...\underline{then}...\underline{else}...$ ought to be written $A_i(x_1,x_2,x_3,x_4,x_5)$ for some i. We shall, however, use the shorthands $x_1=1$ etc. Further we assume that the standard interpretation $\underline{S}$ has the set S to be the integers and that the functions "$x_1=1$" etc. are interpreted as intended. Since there is only one type, truthvalues should be coded by integers, e.g. 1 means true and all other numbers mean false.

From the definition of $\underline{L}$ (and $\underline{I}$) it is straightforward to calculate that

$$\underline{L}(\square)(g_1,g_2) = g_1 \cdot g_2$$

$$\underline{L}(tuple)(g_1,\ldots,g_5) = \lambda(v_1,\ldots,v_5).$$

$$\quad (g_1(v_1,\ldots,v_5),\ \ldots,\ g_5(v_1,\ldots,v_5))$$

$$\underline{L}(take_i) = \lambda(v_1,\ldots,v_5).v_i$$

and for examples of $\underline{L}(a_i)$

$$\underline{L}("x_2+x_3") = \lambda(v_1,\ldots,v_5).\ \min\{v_2,v_3\}$$

$$\underline{L}("\underline{if}\ x_1\ \underline{then}\ x_2\ \underline{else}\ x_3") = \lambda(v_1,\ldots,v_5).\ \min\{v_1,\max\{v_2,v_3\}\}$$

One can then calculate

$$\underline{L}[\![pro]\!] = \lambda(v_1,\ldots,v_5).\min\{v_1,v_2,v_3\}$$

using that the function G of which the least fixed point is required has LFP(G) = G(⊥). The conclusion is that the first three parameters to the program may safely be passed by value. ///

The second analysis we shall consider is ♭ of /Myc81/. It will be formulated as an approximating interpretation $\underline{M}$ and again the elements of $\underline{M}(L_1)$ and $\underline{M}(L_k)$ are $\mathbb{O}$ and $\mathbb{O}^k$ respectively. The intention with 0 now is to denote possible non-termination and with 1 to denote guaranteed termination. So if

$$\underline{M}[\![exp_i]\!](env)(v_1,\ldots,v_k) = 1$$

for some environment env and values v_i then it is safe to use call-by-value for the i'th parameter in the call $F(exp_1,\ldots,exp_i,..,exp_k)$. We refer to /Myc81/ for further information.

The augmented dcpo's $\underline{M}(L_i)$ will be $(\mathbb{O}^i,\leq,\geq)$. The use of $\geq$ is essential in order to achieve $\subseteq$-monotonicity of the concretization functions

$$con_i'' : (\mathbb{O}^i,\leq,\geq) \to \underline{I}(L_i)$$

defined by

$$con_1''(0) = S_\perp \quad , \quad con_1''(1) = S$$

$$con_k''(v_1,\ldots,v_k) = (con_1''(v_1),\ldots,con_1''(v_k))$$

Also note that S is an element of $\mathcal{P}(S_\perp)$ but not (unless S is finite) of the Plotkin powerdomain which was used in /Myc81/. Clearly con_1'' is pseudo-strict and pseudo-continuous and pairs of adjoined functions (abs_1'',con_1'') and (abs_k'',con_k'') are obtained by defining

$$abs_1''(Y) = 1 \text{ if } \perp \notin Y \text{ and } abs_1''(Y) = 0 \text{ otherwise}$$

$$abs_k''(Y_1,\ldots,Y_k) = (abs_1''(Y_1), \ldots,abs_1''(Y_k))$$

Then $\underline{M}$ is defined by

$$\underline{M} = \text{induce}(\underline{I}, (\text{abs}''_i, \text{con}''_i)_i, ((\mathbf{0}^i, \leq, \geq))_i)$$

and as before it is correct with respect to $\underline{C}$.

Example Returning to the example program we may calculate:

$$\underline{M}(\square)(g_1, g_2) = g_1 \dot{} g_2$$

$$\underline{M}(\text{tuple})(g_1, \ldots, g_5) = \lambda(v_1, \ldots, v_5).$$

$$(g_1(v_1, \ldots, v_5), \ldots, g_5(v_1, \ldots, v_5))$$

$$\underline{M}(\text{take}_i) = \lambda(v_1, \ldots, v_5).v_i$$

and for examples of the $\underline{M}(a_i)$:

$$\underline{M}(\text{"}x_2 \times x_3\text{"}) = \lambda(v_1, \ldots, v_5).\ \min\{v_2, v_3\}$$

$$\underline{M}(\text{"}\underline{\text{if}}\ x_1\ \underline{\text{then}}\ x_2\ \underline{\text{else}}\ x_3\text{"}) = \lambda(v_1, \ldots, v_5).\ \min\{v_1, v_2, v_3\}$$

Much as before this allows to calculate that

$$\underline{M}[\![\text{pro}]\!] = \lambda(v_1, \ldots, v_5).0$$

which is clearly correct and in fact as precise as possible. ///

To gain perspective upon this data flow analysis consider the program pro' that is pro with $x_1=1$ replaced by $x_1 \leq 1$. Then $\underline{M}[\![\text{pro}']\!] = \underline{M}[\![\text{pro}]\!]$ which is still correct but not as precise as possible because

$$(\text{abs}''_1 \dot{} \underline{I}[\![\text{pro}']\!] \dot{} \text{con}''_k) = \lambda(v_1, \ldots, v_5).\ \min\{v_1, v_2, v_3\}$$

That is the program terminates if the arguments in the first three positions do but $\underline{M}$ does not detect this. This is not because $\underline{M}$ is badly chosen but more fundamentally because abstract interpretation has only been considered in a "first-order" manner: functions are viewed as transforming (descriptions of) sets of arguments to sets of results and there is no provision for expressing any relationship

between the arguments and the result (e.g. the result is less than the first argument). We shall return to this in the conclusion.

5.6 NONDETERMINISM AS ABSTRACT INTERPRETATION

Nondeterminism means different things in different contexts. In automata theory the nondeterminism chooses between computation paths and preference is given to a path that terminates (in an accept state). This view is closely connected with the amb operator of /McC67/. In programming languages nondeterminism also chooses between computation paths (or values in applicative languages) but all paths are treated equal. This is the notion to be considered here. It is explained in /Egl75/ as: "We do not want to think of trying all possible paths in a program. We are computing along a single path but we might not know which one. This happens, for instance, if the choice of the path is done according to certain unknown parameters in the system which we consider as being nondeterministic choices." We shall use the term oracles for these parameters.

In this section we will show that nondeterminism can be described as a certain abstract interpretation, i.e. as a certain data flow analysis. Such a development has not been performed before and it seems to be essential to use the "strong" approach of this chapter. Furthermore, it will be necessary to extend the framework with the notions of partly collecting and partly induced semantics.

THE EXAMPLE LANGUAGE AND ITS SEMANTICS

The development will be performed for the following language of commands (cmd∈Cmd):

$$\text{cmd} ::= \text{act} \mid \text{cmd}_1 ; \text{cmd}_2 \mid \underline{\text{if}}\ \text{exp}\ \underline{\text{then}}\ \text{cmd}_1\ \underline{\text{else}}\ \text{cmd}_2$$
$$\mid \text{cmd}_1\ \underline{\text{or}}\ \text{cmd}_2 \mid \underline{\text{while}}\ \text{exp}\ \underline{\text{do}}\ \text{cmd}$$

Here act stands for unspecified atomic actions and exp stands for unspecified (boolean) expressions; we shall assume these represent deterministic and total recursive computations. Nondeterminism is introduced by the use of $\text{cmd}_1\ \underline{\text{or}}\ \text{cmd}_2$.

The main idea is to view $\text{cmd}_1\ \underline{\text{or}}\ \text{cmd}_2$ as a shorthand for the following deterministic program

$$\underline{\text{if}}\ \text{hd(o)}\ \underline{\text{then}}\ \text{o:=tl(o)}\ ;\ \text{cmd}_1\ \underline{\text{else}}\ \text{o:=tl(o)}\ ;\ \text{cmd}_2$$

where o is an oracle. One might argue that hd should also have program variables as parameters but we shall not consider this more complicated setting. When the oracle is not known this gives rise to nondeterministic behaviour as in the above quotation from /Egl75/. Therefore denotational semantics (as P below) use powerdomains to collect the sets of possible outcomes. The result of $\text{cmd}_1\ \underline{\text{or}}\ \text{cmd}_2$ is then the union of the results along either branch. This is quite analogous to what happens in the collecting (or another approximating) semantics if too little is known about the set of states so that the conditional may evaluate to both true and false. In the remainder of this section this idea will be precisely formulated using abstract interpretation.

We first define a denotational semantics P for the nondeterministic language. Let S be an unspecified nonempty set of states. The meaning of atomic actions and (boolean) expressions are given by unspecified total functions

$$\mathcal{A}[\![\text{act}]\!] : S \to S$$
$$\mathcal{B}[\![\text{exp}]\!] : S \to \{\text{true},\text{false}\}$$

We shall extend these strictly to $S_\perp \to S_\perp$ and $S_\perp \to \{true, false\}_\perp$ respectively. The nondeterministic semantics

$$\underline{P}[\![cmd]\!] : S_\perp \to \mathcal{P}(S_\perp)$$

is then defined as follows (using notation from section 5.2):

$$\underline{P}[\![act]\!] = \{\!|\cdot \mathcal{A}[\![act]\!]$$

$$\underline{P}[\![cmd_1 ; cmd_2]\!] = \underline{P}[\![cmd_2]\!] \,\square^P\, \underline{P}[\![cmd_1]\!]$$

$$\underline{\text{where}}\ f_2 \,\square^P\, f_1 = f_2^\dagger \cdot f_1$$

$$\underline{P}[\![\underline{\text{if}}\ exp\ \underline{\text{then}}\ cmd_1\ \underline{\text{else}}\ cmd_2]\!] = cond^P_{exp}(\underline{P}[\![cmd_1]\!], \underline{P}[\![cmd_2]\!])$$

$$\underline{\text{where}}\ cond^P_{exp}(f_1, f_2)(s) = \mathcal{B}[\![exp]\!](s) \to f_1(s),\ f_2(s)$$

$$\underline{P}[\![cmd_1\ \underline{or}\ cmd_2]\!] = \underline{P}[\![cmd_1]\!]\ or^P\ \underline{P}[\![cmd_2]\!]$$

$$\underline{\text{where}}\ (f_1\ or^P\ f_2)(s) = f_1(s) \cup f_2(s)$$

$$\underline{P}[\![\underline{\text{while}}\ exp\ \underline{do}\ cmd]\!] = LFP(\lambda f.\ cond^P_{exp}(\ f \,\square^P\, \underline{P}[\![cmd]\!]\ ,\ \{\}))$$

This defines a strict monotonic function of functionality as stated. Because $S_\perp$ is of finite height it is (directed) continuous as well. Usually the Plotkin powerdomain is used instead of $\mathcal{P}(S_\perp)$ but this does not affect the sets produced by the semantics. (As in /Plo76/ an argument using König's lemma shows that $\underline{P}[\![cmd]\!](s)$ is either finite or contains $\perp$.)

To obtain the above semantics as an approximating semantics we must start with a deterministic standard semantics. It will use oracles modelled by infinite strings of 0's and 1's. (We shall discuss this choice further in the next section.) The idea then is to define a semantics

$$\underline{S}[\![cmd]\!] : S_\perp * O_\perp \to S_\perp * O_\perp$$

where $O = \{0,1\}^\omega$ is the set of countably infinite strings of 0's and 1's. When a "nondeterministic branch" is encountered the left branch

will be taken if the oracle begins with 0 and the right branch if the oracle begins with 1. The branch chosen is then executed with the remainder of the oracle. This in effect corresponds to viewing cmd_1 or cmd_2 as a shorthand for the piece of program displayed earlier. As for the functionality of $\underline{S}[\![cmd]\!]$ it is possible to use $S_\perp \times O_\perp$, but $S_\perp \to O_\perp$ is closer to "operational intuitions" and this is preferable in the next section.

The following operations upon oracles $o \in O$ and finite strings $w \in \{0,1\}^*$ will be needed. By $o\upharpoonright m$ and $o\dagger m$ are denoted the first m elements of o and the remainder of o respectively. We shall extend this notation to $O_\perp$ by writing $\perp\upharpoonright m = \perp = \perp\dagger m$. The string w has length $|w|$ and wo is the oracle obtained by prefixing w to o. The standard semantics $\underline{S}$ is then defined by the following equations:

$$\underline{S}[\![act]\!] = \lambda(s,o).\mathcal{A}[\![act]\!](s),o)$$

$$\underline{S}[\![cmd_1;cmd_2]\!] = \underline{S}[\![cmd_2]\!] \cdot \underline{S}[\![cmd_1]\!]$$

$$\underline{S}[\![\underline{if}\ exp\ \underline{then}\ cmd_1\ \underline{else}\ cmd_2]\!] = cond^S_{exp}(\underline{S}[\![cmd_1]\!], \underline{S}[\![cmd_2]\!])$$

$$\underline{where}\ cond^S_{exp}(f_1,f_2)(s,o) = \mathcal{B}[\![exp]\!](s) \to f_1(s,o),\ f_2(s,o)$$

$$\underline{S}[\![cmd_1\ \underline{or}\ cmd_2]\!] = \underline{S}[\![cmd_1]\!]\ or^S\ \underline{S}[\![cmd_2]\!]$$

$$\underline{where}\ (f_1\ or^S\ f_2)(s,o) = (o\upharpoonright 1=0) \to f_1(s,o\dagger 1),\ f_2(s,o\dagger 1)$$

$$\underline{S}[\![\underline{while}\ exp\ \underline{do}\ cmd]\!] = LFP(\lambda f.\ cond^S_{exp}(f \cdot \underline{S}[\![cmd]\!], \lambda(s,o).(s,o)))$$

These equations define a strict function of functionality as stated. For O is chosen such that $o\dagger 1 \in O$ whenever $o \in O$ and since $\mathcal{A}[\![act]\!]$ is a strict extension of a total function we know that $\underline{S}[\![act]\!]$ is of the correct functionality. In fact $\underline{S}[\![cmd]\!]$ is continuous and it is straightforward to show that LFP is only needed of continuous functionals. It is also easy to show that the semantics is "continuous" in prefixes of oracles:

Fact 5.6:1. If $\underline{S}[\![cmd]\!](s,o) = (s',o') \neq (\perp,\perp)$ there exists a finite prefix w of o such that $o"\Gamma|w| = w$ implies that $\underline{S}[\![cmd]\!](s,o") = (s',o"\Gamma|w|)$. ///

THE PARTLY COLLECTING SEMANTICS

The next task is to define a collecting semantics and show its relation to the standard semantics. One might consider

$$\underline{C}[\![cmd]\!] : \mathcal{P}(S_\perp * O_\perp) \to \mathcal{P}(S_\perp * O_\perp)$$

as this follows the pattern used previously but it will be more appropriate to use

$$\underline{C}[\![cmd]\!] : S_\perp \times \mathcal{P}(O_\perp) \to \mathcal{P}(S_\perp * O_\perp)$$

which will be called the partly collecting semantics. This is because the idea with the nondeterministic semantics is that we have full information about the states and no information about the oracles. In other words data flow analysis is performed only for the oracles and not for the states. (It should be intuitively clear that $\mathcal{P}(S_\perp * O_\perp)$ in the range cannot be replaced by $S_\perp \times \mathcal{P}(O_\perp)$ because oracles may influence the resulting states.)

To define $\underline{C}$ we first consider the necessary modifications of theorem 5.2:5. Let D be a dcpo and B and C augmented dcpo's. A function $g:D\times B\to C$ is $\subseteq$-monotonic in its right argument (ambiguously written $g:D\times B\xrightarrow[\subseteq]{}C$) iff for all $d\in D$ that $Y_1\subseteq Y_2$ implies $g(d,Y_1)\subseteq g(d,Y_2)$. If the augmentations of B and C are complete the function is said to be completely linear in its right argument (written $g:D\times B\xrightarrow[\cup]{}C$) iff, for each $d\in D$ and nonempty subset $\mathcal{Y}$ of B, the formula

$$g(d,\cup\mathcal{Y}) = \cup\{g(d,Y) \mid Y\in\mathcal{Y}\}$$

holds. The function is smash-invariant ($g:D\times B\to_{*} C$) iff $g=g^{\cdot}smash$. Finally, let $\tilde{f}:D\times E\to D\times F$ denote the application of $f:E\to F$ upon second components, i.e. $\tilde{f}(d,e) = (d,f(e))$ and let $inc:D\times B\to D\times B$ be defined by $inc(d,Y) = (d,Y)$.

Lemma 5.6:2. Let D be a dcpo, E a benign fdcpo and C a completely augmented dcpo. For each monotonic function $f:D\times E\to C$ there exists precisely one monotonic and smash-invariant function $f^{\#}:D\times\wp(E)\to C$ that is completely linear in its right argument and fulfils that $f^{\#}\cdot\tilde{\{\}}\cdot inc = f$. It is given by the formula

$$f^{\#}(d,Y) = \bigcup\{f(smash(d,y)) \mid y\in Y\}$$

and gives a strict function iff f is strict. Further $\lambda f.f^{\#}$ is a monotonic and $\subseteq$-monotonic bijection from $D\times E\to C$ to $D\times\wp(E)\to_{*,*} C$ whose inverse $\lambda g.g\cdot\tilde{\{\}}\cdot inc$ is also monotonic and $\subseteq$-monotonic. ///

Proof The proof is similar to that of theorem 5.2:5 so the details are omitted. ///

The main use of this lemma is to "lift" a function $f:D\times E\to F$ to a function $\hat{f}:D\times\wp(E)\to_{\subseteq}\wp(F)$ akin to what was done when extending $\wp$ to a functor. For this let D be a dcpo and E and F benign fdcpo's. Define

$$\hat{f} = (\{\}\cdot f)^{\#} = \lambda(d,Y).\ CON(\{f(smash(d,y)) \mid y\in Y\})$$

Clearly $\hat{f}$ is monotonic, smash-invariant and completely linear (hence $\subseteq$-monotonic) in its right argument and it is strict iff f is. We shall also need to define $\bar{g}$ akin to LIN(g) defined previously. For this let C be a completely augmented dcpo and $g:D\times\wp(E)\to C$ a monotonic function and define $\bar{g}:D\times\wp(E)\to_{\subseteq} C$ by

$$\bar{g} = (g \cdot \widetilde{\{\}} \cdot inc)^{\#} = \lambda(d,Y).\ \mathcal{U}\{g(smash(d,\{y\})) \mid y \in Y\}$$

It is immediate that $\bar{g}$ is smash-invariant and is completely linear (hence $\subseteq$-monotonic) in its right argument. It is strict iff g is.

We are now ready to define the partly collecting semantics

$$\underline{C}[\![cmd]\!] : S_\perp \times \mathcal{P}(O_\perp) \to_{\subseteq} \mathcal{P}(S_\perp * O_\perp)$$

The equations are:

$$\underline{C}[\![act]\!] = \overbrace{\lambda(s,o).\ (\mathcal{A}[\![act]\!](s),o)}$$

$$\underline{C}[\![cmd_1;cmd_2]\!] = \underline{C}[\![cmd_2]\!] \ \square^C \ \underline{C}[\![cmd_1]\!]$$

$$\underline{\text{where}}\ g_2 \ \square^C \ g_1 = (g_2 \cdot \widetilde{\{\}} \cdot inc)^\dagger \cdot g_1$$

$$\underline{C}[\![\underline{\text{if}}\ exp\ \underline{\text{then}}\ cmd_1\ \underline{\text{else}}\ cmd_2]\!] = cond^C_{exp}(\underline{C}[\![cmd_1]\!], \underline{C}[\![cmd_2]\!])$$

$$\underline{\text{where}}\ cond^C_{exp}(g_1,g_2)(s,X) = \mathcal{B}[\![exp]\!](s) \to g_1(s,X),\ g_2(s,X)$$

$$\underline{C}[\![cmd_1\ \underline{\text{or}}\ cmd_2]\!] = \underline{C}[\![cmd_1]\!]\ \ or^C\ \underline{C}[\![cmd_2]\!]$$

$$\underline{\text{where}}\ (g_1\ or^C\ g_2)(s,X) = g_1(s,\{o\dagger 1 \mid o \in X \wedge o\dagger 1 = 0\})$$
$$g_2(s,\{o\dagger 1 \mid o \in X \wedge o\dagger 1 = 1\})$$
$$\{(\perp,\perp) \mid \perp \in X\}$$

with the convention that $g_i(s,\emptyset)=\emptyset$ and that $\emptyset \cup Z = Z$

$$\underline{C}[\![\underline{\text{while}}\ exp\ \underline{\text{do}}\ cmd]\!] = LFP(\lambda g.\ cond^C_{exp}(\ g \ \square^C \ \underline{C}[\![cmd]\!],\ \overbrace{\lambda(s,o).(s,o)}))$$

It is straightforward to verify that this defines a strict and monotonic function that is $\subseteq$-monotonic in its right argument. Because $\subseteq$-monotonicity in the right argument is an admissible property the least upper bounds implicit in the definition of LFP are pointwise.

The connection with the standard semantics is given by:

Theorem 5.6:3. $\underline{C}[\![cmd]\!] = \overbrace{\underline{S}[\![cmd]\!]} = \lambda(s,X).\ \{\underline{S}[\![cmd]\!](smash(s,o)) \mid o \in X\}$ ///

Proof The proof follows the pattern used in proving lemma 5.3:2 and theorem 5.3:3.

Define the predicate P(f,g) by $g \cdot \{\!|\}\!|^{\sim} \cdot inc = \{\} \cdot f$. We first show by structural induction on commands cmd that $P(\underline{S}[\![cmd]\!], \underline{C}[\![cmd]\!])$ holds. The result is immediate for atomic actions. Suppose next it holds for cmd_1 and cmd_2. For $cmd = cmd_1 ; cmd_2$ calculate

$$\underline{C}[\![cmd]\!] \cdot \{\!|\}\!|^{\sim} \cdot inc =$$

(using the definition of $\underline{C}[\![cmd]\!]$ and the induction hypotheses)

$$(\{\} \cdot \underline{S}[\![cmd_2]\!])^{\dagger} \cdot \{\} \cdot \underline{S}[\![cmd_1]\!] =$$

(using theorem 5.2:5 and the definition of $\underline{S}[\![cmd]\!]$)

$$\{\} \cdot \underline{S}[\![cmd]\!]$$

For cmd = <u>if</u> exp <u>then</u> cmd_1 <u>else</u> cmd_2 it easily follows by doing case analysis upon $\mathcal{B}[\![exp]\!](s)$ that

$$\underline{C}[\![cmd]\!](\{\!|\}\!|^{\sim}(inc(s,o))) = \{\underline{S}[\![cmd]\!](s,o)\}$$

holds for all $(s,o) \in S_\perp * O_\perp$. For cmd = cmd_1 <u>or</u> cmd_2 a similar result follows by doing case analysis upon $o|_1$. Finally consider cmd = <u>while</u> exp <u>do</u> cmd_1 and let F and G be the functionals of which the least fixed points are required (for <u>S</u> and <u>C</u> respectively). Reasoning as above it is straightforward to show that

$$P(f,g) \text{ implies } P(F(f),G(g))$$

The result then follows by the induction principle once it is shown that P is admissible. It is immediate that $P(\perp,\perp)$ holds and for the other condition note that $\{\}$ is directed continuous and the least upper bounds (of directed set) in $S_\perp \times \mathcal{C}(O_\perp) \to_c \mathcal{E}(S_\perp * O_\perp)$ are pointwise.

To obtain the result of the theorem it suffices to show that $\underline{C}[\![cmd]\!]$ is completely linear in its right argument. This is shown by a straightforward structural induction. Taking <u>while</u> exp <u>do</u> cmd as an example it is immediate that G(g) is always completely linear

in its right argument. Hence the result follows for LFP(G)=G(LFP(G)).

///

Corollary 5.6:4. For G as defined in the above proof we have

$$LFP(G) = \overline{\bigsqcup_{n=0} G^n(\bot)}$$ ///

Proof Using F defined above we calculate

$$LFP(G) =$$

$$\widehat{LFP(F)} =$$

(F and $\{\!|\}$ are continuous)

$$(\bigsqcup_n \{\!|\}\cdot F^n(\bot))^{\#} =$$

(a straightforward numerical induction)

$$(\bigsqcup_n G^n(\bot)\cdot\{\!|\}\cdot \mathrm{inc})^{\#} =$$

($\subseteq$-monotonicity is admissible so the least upper bound is pointwise)

$$\overline{((\bigsqcup_n G^n(\bot))\cdot\{\!|\}\cdot \mathrm{inc})^{\#}} =$$

$$\overline{\bigsqcup_n G^n(\bot)}$$

as was to be shown. ///

THE PARTLY INDUCED SEMANTICS

To obtain the nondeterministic semantics the idea is to replace $\mathcal{P}(O_\bot)$ and $\mathcal{P}(S_\bot \times O_\bot)$ by less informative augmented dcpo's B and C respectively. This amounts to defining a partly induced semantics

$$\underline{I}[\![\mathrm{cmd}]\!] : S_\bot \times B \to_{\subseteq} C$$

The connection between these augmented dcpo's is given by the following pairs of abstraction and concretization functions

$$(\mathrm{abs}: \mathcal{P}(O_\bot) \to B,\ \mathrm{con}: B \to \mathcal{P}(O_\bot))$$

$$(\mathrm{abs}': \mathcal{P}(S_\bot \times O_\bot) \to C,\ \mathrm{con}': C \to \mathcal{P}(S_\bot \times O_\bot))$$

We shall assume that all these functions are monotonic and $\subseteq$-monotonic

(but adjoined-ness will fail as will be seen). The partly induced semantics is then defined by the following equations:

$$\underline{I}[\![act]\!] = abs'' \cdot \overbrace{\lambda(s,o).(\mathcal{A}[\![act]\!](s),o)} \cdot \widetilde{con}$$

$$\underline{I}[\![cmd_1;cmd_2]\!] = \underline{I}[\![cmd_2]\!] \; \square^I \; \underline{I}[\![cmd_1]\!]$$

$$\underline{\text{where}}\ h_2 \; \square^I \; h_1 = abs'' \cdot ((con'' \cdot h_2 \cdot \widetilde{abs}) \; \square^C \; (con'' \cdot h_1 \cdot \widetilde{abs})) \cdot \widetilde{con}$$

$$\underline{I}[\![\underline{\text{if}}\ exp\ \underline{\text{then}}\ cmd_1\ \underline{\text{else}}\ cmd_2]\!] = cond^I_{exp}(\underline{I}[\![cmd_1]\!], \underline{I}[\![cmd_2]\!])$$

$$\underline{\text{where}}\ cond^I_{exp}(h_1,h_2) = abs'' \cdot cond^C_{exp}(con'' \cdot h_1 \cdot \widetilde{abs}, \; con'' \cdot h_2 \cdot \widetilde{abs}) \cdot \widetilde{con}$$

$$\underline{I}[\![cmd_1\ \underline{\text{or}}\ cmd_2]\!] = \underline{I}[\![cmd_1]\!] \; or^I \; \underline{I}[\![cmd_2]\!]$$

$$\underline{\text{where}}\ h_1\ or^I\ h_2 = abs'' \cdot ((con'' \cdot h_1 \cdot \widetilde{abs})\ or^C\ (con'' \cdot h_2 \cdot \widetilde{abs})) \cdot \widetilde{con}$$

$$\underline{I}[\![\underline{\text{while}}\ exp\ \underline{\text{do}}\ cmd]\!] = LFP(\lambda h. \; cond^I_{exp}(h \; \square^I \; \underline{I}[\![cmd]\!], \; abs'' \cdot \overbrace{\lambda(s,o).(s,o)} \cdot \widetilde{con}))$$

We shall assume that the augmentation of C is admissible. Then $\subseteq$-monotonicity in the right argument is an admissible property and therefore $S_\perp \times B \to_{\subseteq} C$ is a dcpo. It is then straightforward to verify that the equations define a monotonic function that is $\subseteq$-monotonic in its right argument.

To obtain the nondeterministic semantics we define $B = (\{\perp\}, \subseteq, \in)$ with $abs(X) = \perp$ and $con(\perp) = O$ as well as $C = \mathcal{P}(S_\perp)$ with

$$abs'(Z) = \{s \mid \exists o: (s,o) \in Z\}$$

$$con'(Y) = \{smash(s,o) \mid s \in Y \wedge o \in O\}$$

We shall let $\underline{I}[\![cmd]\!]$ refer to the partly induced semantics when B etc. are as above. Because O is not empty it is immediate to verify that the functions are monotonic and $\subseteq$-monotonic. In fact (abs',con') is a pair of exactly adjoined functions so abs' is completely linear. But (abs,con) is not a pair of adjoined functions because although $abs \cdot con = \lambda 1.1$ the other condition fails.

Remark One way to view the failure of adjoined-ness is that the choice of B accomplishes two goals rather than one. The first goal is to replace $\mathcal{P}(O_\perp)$ by a less informative structure and

$$(\{O, O_\perp\}, \sqsubseteq_{EM}, \subseteq)$$

might be a suitable choice. The second goal is to remove unneeded elements (i.e. $O_\perp$) from B in order to come close to the nondeterministic semantics. It therefore might be considered to accomplish these two goals in two seperate steps but this is not needed to demonstrate the connection between nondeterminism and abstract interpretation. ///

Clearly $S_\perp \times B$ and B are isomorphic so we shall subsequently identify them. We then have:

Theorem 5.6:5. $\underline{I}[\![cmd]\!] = \underline{P}[\![cmd]\!]$ for all commands cmd. ///

Proof The proof is by structural induction upon cmd.

Case cmd = act. We have

$$\underline{I}[\![act]\!](s) =$$

$$abs'(\{(\mathcal{A}[\![act]\!](s), o) \mid o \in O\}) =$$

$$\{\mathcal{A}[\![act]\!](s)\} =$$

$$\underline{P}[\![act]\!](s)$$

Case cmd = $cmd_1; cmd_2$. We have

$$\underline{I}[\![cmd]\!] =$$

$$\underline{P}[\![cmd_2]\!] \natural^I \underline{P}[\![cmd_1]\!] =$$

$$abs'' \cdot (con'' \cdot \underline{P}[\![cmd_2]\!] \cdot \widetilde{abs} \cdot \{\} \cdot inc)^\dagger \cdot con'' \cdot \underline{P}[\![cmd_1]\!] \cdot \widetilde{abs} \cdot \widetilde{con} =$$

(by abs' completely linear and $abs'' \cdot con'$ and $\widetilde{abs} \cdot \widetilde{con}$ the identities)

$$(\underline{P}[\![cmd_2]\!] \cdot \widetilde{abs} \cdot \{\} \cdot inc)^\dagger \cdot con'' \cdot \underline{P}[\![cmd_1]\!] =$$

(by $(h \cdot \widetilde{abs} \cdot \{\} \cdot inc)^\dagger = h^\dagger \cdot abs'$ as is straightforward to show)

$$\underline{P}[\![cmd_2]\!]^\dagger \cdot abs'' \cdot con'' \cdot \underline{P}[\![cmd_1]\!] = \underline{P}[\![cmd]\!]$$

Case cmd = if exp then cmd_1 else cmd_2. We have

$$\underline{I}[\![cmd]\!] =$$

$$abs'' \cdot cond^{C}_{exp}(con'' \cdot \underline{P}[\![cmd_1]\!] \cdot \widetilde{abs}, \ldots) \cdot \widetilde{con} =$$

(by abs' strict)

$$cond^{P}_{exp}(abs'' \cdot con'' \cdot \underline{P}[\![cmd_1]\!] \cdot \widetilde{abs} \cdot \widetilde{con}, \ldots) =$$

$$cond^{P}_{exp}(\underline{P}[\![cmd_1]\!], \underline{P}[\![cmd_2]\!]) =$$

$$\underline{P}[\![cmd]\!]$$

Case cmd = cmd_1 or cmd_2. We have

$$\underline{I}[\![cmd]\!] =$$

$$abs'' \cdot ((con'' \cdot \underline{P}[\![cmd_1]\!] \cdot \widetilde{abs}) \; or^{C} \ldots) \cdot \widetilde{con} =$$

(using the convention described when defining or^{C} and the complete linearity of abs')

$$\lambda s.\ (abs'' \cdot con'' \cdot \underline{P}[\![cmd_1]\!] \cdot \widetilde{abs})(s, \{o \dagger 1 \mid o \in O \wedge o\Gamma 1 = 0\}) \cup \ldots \cup \{\perp \mid \perp \in O\} =$$

(because there are $o', o'' \in O$ such that $o'\Gamma 1 = 0$ and $o''\Gamma 1 = 1$ and because $\perp \notin O$)

$$\lambda s.\ \underline{P}[\![cmd_1]\!](s) \cup \underline{P}[\![cmd_2]\!](s) =$$

$$\underline{P}[\![cmd]\!]$$

Case cmd = while exp do cmd_1. Let $\underline{P}[\![cmd]\!] = LFP(F)$ and $\underline{I}[\![cmd]\!] = LFP(H)$. Reasoning as above it follows that $F(f) = H(h)$ whenever $f = h$. The result then follows by the induction principle of section 5.1 because equality is an admissible predicate. ///

The following theorem establishes the final result needed to verify the claim that the nondeterministic semantics essentially is the result of performing an abstract interpretation upon the deterministic standard semantics. So far it has not been essential that $O = \{0,1\}^{\omega}$ and the development could have been performed for any subset O of $\{0,1\}^{\omega}$ such that $o \dagger 1$ is an element of O whenever o is and O contains

oracles of the form 0o' and 1o". In the proof below more properties of O will be used.

Theorem 5.6:6. $\underline{I}[\![cmd]\!] = abs'' \cdot \underline{C}[\![cmd]\!] \cdot \widetilde{con}$ holds for all commands cmd.

///

Proof We begin with defining the following predicate P that is central to the proof. It is defined upon

$$(S_\perp \times \mathcal{P}_\omega(O_\perp) \to_{\subseteq} \mathcal{P}_\omega(S_\perp \times O_\perp)) \times (S_\perp \times B \to_{\subseteq} C)$$

and P(g,h) is defined to hold iff all of

(a) $h = abs'' \cdot g \cdot con$

(b) $g: S_\perp \times \mathcal{P}_\omega(O_\perp) \xrightarrow[s\sharp\mathcal{P}]{} \mathcal{P}_\omega(S_\perp \times O_\perp)$

(c) for all $s \in S_\perp$: $g(s,O) = \{smash(s',o) \mid o \in O \wedge \exists o' : (s',o') \in g(s,O)\}$

hold. Condition (a) is the desired result but (b) and (c) will be needed to prove it. When $g = \underline{C}[\![cmd]\!]$ it is immediate from theorem 5.6:3 and the observations after lemma 5.6:2 that condition (b) holds. We shall see that (c) holds as well. It may be phrased as saying that whenever one oracle is possible then all are. An equivalent formulation of (c) is that the range of $g \cdot \widetilde{con}$ is a subset of the range of con'.

The proof proceeds by structural induction on cmd and shows

$$P(\underline{C}[\![cmd]\!], \underline{I}[\![cmd]\!])$$

Case cmd = act is straightforward.

Case cmd = $cmd_1; cmd_2$. This amounts to assuming that $P(g_1,h_1)$ and $P(g_2,h_2)$ hold and then show that $P(g_2 \,\square^C\, g_1,\ h_2 \,\square^I\, h_1)$ holds. For (a) calculate

$$h_2 \ \square^I \ h_1 =$$

(as in the proof of theorem 5.6:5)

$$h_2^{\dagger} \cdot h_1 =$$

$$(abs'' \cdot g_2 \cdot \widetilde{con})^{\dagger} \cdot abs'' \cdot g_1 \cdot \widetilde{con} =$$

(abs' completely linear and $h^{\dagger} \cdot abs' = (h \cdot \widetilde{abs} \cdot \widetilde{\{\}} \cdot inc)^{\dagger}$)

$$abs'' \cdot (g_2 \cdot \widetilde{con} \cdot \widetilde{abs} \cdot \widetilde{\{\}} \cdot inc)^{\dagger} \cdot g_1 \cdot \widetilde{con} =$$

(to be argued below)

$$abs'' \cdot (g_2 \cdot \widetilde{\{\}} \cdot inc)^{\dagger} \cdot g_1 \cdot \widetilde{con} =$$

$$abs'' \cdot (g_2 \ \square^c \ g_1) \cdot \widetilde{con} =$$

For the missing step it suffices to show that

$$(g_2 \cdot \widetilde{con \cdot abs} \cdot \widetilde{\{\}} \cdot inc)^{\dagger} \cdot con' = (g_2 \cdot \widetilde{\{\}} \cdot inc)^{\dagger} \cdot con'$$

because g_1 satisfies condition (c), i.e. $g_1(\widetilde{con}(s,\perp))$ equals some con'(Y). Abbreviate this equation to LHS = RHS. By complete linearity it suffices to consider singletons. We have

$$LHS(\{s\}) = g_2(s,0)$$

$$RHS(\{s\}) = \cup\{g_2(s,\{o\}) \mid o \in O\}$$

and these are equal because g_2 satisfies (b).

It is straightforward to verify condition (b). For condition (c) it follows from the calculations above that

$$(g_2 \ \square^c \ g_1)(\widetilde{con}(s,\perp)) =$$

$$LHS(Y) = \qquad \text{(for some Y)}$$

$$\cup\{(g_2 \cdot \widetilde{con})(s,\perp) \mid s \in Y\} =$$

$$\cup\{con'(Y_s) \mid s \in Y\} = \qquad \text{(for some } Y_s\text{)}$$

$$con'(\cup\{Y_s \mid s \in Y\})$$

This shows condition (c).

Case cmd = if exp then cmd_1 else cmd_2. This amounts to assuming that

$P(g_1,h_1)$ and $P(g_2,h_2)$ hold and then show that

$P(\ cond^C_{exp}(g_1,g_2)\ ,\ cond^I_{exp}(h_1,h_2)\ )$ holds. The calculations for (a) are

$$cond^I_{exp}(h_1,h_2) =$$

(as in the proof of theorem 5.6:5)

$$\lambda(s,\perp).\ \mathcal{B}[\![exp]\!](s) \to h_1(s,\perp),\ h_2(s,\perp) =$$

$$abs''(\lambda(s,X).\mathcal{B}[\![exp]\!](s)\to g_1(s,X),\ g_2(s,X))\cdot\widetilde{con} =$$

$$abs''cond^C_{exp}(g_1,g_2)\cdot\widetilde{con}$$

Verification of conditions (b) and (c) are straightforward.

Case cmd = cmd_1 or cmd_2. This amounts to assuming that $P(g_1,h_1)$ and $P(g_2,h_2)$ hold and show that $P(g_1\ or^C\ g_2,\ h_1\ or^I\ h_2)$ holds. For (a) we calculate

$$h_1\ or^I\ h_2 =$$

(as in the proof of theorem 5.6:5)

$$\lambda(s,\perp).\ (abs''g_1\cdot\widetilde{con})(s,\perp)\ \cup\ (abs''g_2\cdot\widetilde{con})(s,\perp) =$$

$$abs''(\lambda(s,X).g_1(s,X)\cup g_2(s,X))\cdot\widetilde{con} =$$

(because $\{\sigma\tau 1 \mid \sigma\in 0 \wedge \sigma\lceil 1=j\}$ equals 0 for j=0 and j=1)

$$abs''(g_1\ or^C\ g_2)\cdot\widetilde{con}$$

Condition (b) is straightforward to verify. For condition (c) it follows from above that $(g_1\ or^C\ g_2)\cdot\widetilde{con}$ equals

$$\lambda(s,\perp).\ (g_1\cdot\widetilde{con})(s,\perp)\ \cup\ (g_2\cdot\widetilde{con})(s,\perp)$$

and by the assumptions about g_i condition (c) easily follows.

Case cmd = while exp do cmd_1. Let F, G and H be the functionals such that $\underline{S}[\![cmd]\!]$ = LFP(F), $\underline{C}[\![cmd]\!]$ = LFP(G) and $\underline{I}[\![cmd]\!]$ = LFP(H). It follows from the cases above that

$$P(g,h) \text{ implies } P(G(g),H(h)) \tag{1}$$

We first show that P(LFP(G) , LFP(H)) follows from

$$\overline{\bigsqcup_{n=0}^{\infty} G^n(\bot)} \cdot \widetilde{con} = con' \cdot \bigsqcup_{n=0}^{\infty} H^n(\bot) \tag{2}$$

which we shall abbreviate to LHS = RHS. First, condition (a) of

$$P(\overline{\bigsqcup_n G^n(\bot)}, \bigsqcup_n H^n(\bot))$$

follows because abs''con' is the identity. Condition (b) is immediate by the properties of $\lambda g.\bar{g}$ and condition (c) is immediate. Next P(LFP(G) , $\bigsqcup_n H^n(\bot)$) follows because of corollary 5.6:4. Using (1) this shows that

$$H(\bigsqcup_n H^n(\bot)) = abs' \cdot LFP(G) \cdot \widetilde{con} = \bigsqcup_n H^n(\bot)$$

and therefore $\bigsqcup_n H^n(\bot)$ equals LFP(H). Hence P(LFP(G),LFP(H)) follows from (2).

To establish (2) we first rewrite

$$RHS = \bigsqcup_n con' \cdot H^n(\bot)$$

To prove this it suffices to show that con' is directed continuous. Clearly it is monotonic and completely linear. This implies continuity because $S_\bot$ and $S_\bot * O_\bot$ are flat. To see this let $\mathcal{Y}$ be a directed subset of $\mathcal{P}(S_\bot)$. If all $Y \in \mathcal{Y}$ contain $\bot$ then all con'(Y) contain $(\bot,\bot)$ and

$$con'(\bigsqcup \mathcal{Y}) = con'(\bigcup \mathcal{Y}) = \bigcup\{con'(Y) \mid Y \in \mathcal{Y}\} = \bigsqcup\{con'(Y) \mid Y \in \mathcal{Y}\}$$

Otherwise $\bigsqcup \mathcal{Y}$ is an element of $\mathcal{Y}$ and continuity follows by monotonicity.

Both LHS and RHS are strict functions so it suffices to fix $s \in S$ and show LHS(s,$\bot$) = RHS(s,$\bot$). We have, using the connection between F and G, that

$$LHS(s,\perp) =$$

$$\cup\{\{\sqcup_n F^n(\perp)(s,o)\} \mid o\in O\} =$$

$$\{\sqcup_n F^n(\perp)(s,o) \mid o\in O\}$$

Since (1) easily gives $P(G^n(\perp),H^n(\perp))$ we have

$$RHS(s,\perp) =$$

$$\sqcup_n G^n(\perp)(s,O) =$$

$$\sqcup_n \cup\{G^n(\perp)(s,\{o\}) \mid o\in O\} =$$

$$\sqcup_n \{F^n(\perp)(s,o) \mid o\in O\}$$

It is straightforward to show that $LHS(s,\perp)$ and $RHS(s,\perp)$ contain the same maximal elements (i.e. elements that are not $(\perp,\perp)$). It is also straightforward to show that if $LHS(s,\perp)$ contains $(\perp,\perp)$ then so does $RHS(s,\perp)$.

Next assume that $RHS(s,\perp)$ contains $(\perp,\perp)$ and show that $LHS(s,\perp)$ also does. From the assumption it follows that for each n there is $o_n\in O$ such that $F^n(\perp)(s,o_n) = (\perp,\perp)$. We must find $o\in O$ such that

$$\sqcup_m F^m(\perp)(s,o) = (\perp,\perp)$$

If no o_n can be used for o we construct o as follows.

For each n there is a minimal number m_n such that $F^{m_n}(\perp)(s,o_n)\neq(\perp,\perp)$. It follows that $m_n>n$ and it is not hard to show that this implies that

$$F^n(f)(s,o_n) = f(\underline{S}[\![cmd_1]\!]^n(s,o_n))$$

Since $f = F^{m_n-n}(\perp)$ is strict it follows that $\underline{S}[\![cmd_1]\!]^n(s,o_n) \neq (\perp,\perp)$. By n applications of fact 5.6:1 this gives k_n such that

$$o\restriction k_n = o_n\restriction k_n \text{ implies } F^n(\perp)(s,o) = \perp$$

The set $\{o_n\restriction k_n \mid n\geq 0\}$ must be infinite. For if it was finite there would be some n such that $o_j\restriction k_j = o_n\restriction k_n$ holds for infinitely many j.

Then

$$F^j(\perp)(s,o_n) = (\perp,\perp)$$

would hold for infinitely many j and this contradicts the assumption that no o_n could be used for o. One may view the set $\{o_n\restriction k_n \mid n\geq 0\}$ as a tree ordered by prefix. It is an infinite but finitely branching tree. By König's lemma there must exist an infinite path $o\in\{0,1\}^\omega$. By choice of O we have $o\in O$. Since $o\restriction k_m = o_j\restriction k_m$ holds for infinitely many m we have that $\bigsqcup_m F^m(\perp)(s,o) = (\perp,\perp)$. ///

5.7 OTHER NOTIONS OF NONDETERMINISM

In the previous section it was assumed that <u>all</u> countably infinite strings of 0's and 1's were possible oracles. From a data flow analysis point of view it is natural to investigate what happens if only a certain set of strings is possible. For example if "the choice of the path is done according to certain unknown parameters in the system"/Egl75/ one might want to consider only "computable" strings. Similarly fairness considerations may restrict attention to only "fair" strings. This section investigates consequences of letting the set O of oracles be a subset of $\{0,1\}^\omega$.

First we consider what properties O is to fulfil in order that the results of the previous section still hold and a partial answer is given. Next there is the problem of how to formulate a nondeterministic semantics, i.e. a variant of <u>P</u>, such that it corresponds to the effect of some chosen set O of oracles. It turns out that for some choices of O there is no modification of <u>P</u> (in a certain class of possibilities) that yields the desired semantics. This class contains the partly induced semantics and therefore abstract inter-

pretation (as developed in section 5.6) is of no help in this. It is therefore studied how to obtain a modification A of P such that the desired semantics will be "between" P and A: it converges if P says so and diverges if A says so. In particular, whereas P specifies divergence if the program may diverge A only does so if the program may have to diverge. For this reason A is called the "angelic semantics".

On replacing $\{0,1\}^\omega$ by an "equivalent" subset

Let O be a subset of $\{0,1\}^\omega$. Throughout this section we shall assume that O satisfies the following acceptability conditions:

$$O \neq \emptyset$$

$$O = \{o\dagger 1 \mid o \in O \wedge o\lceil 1 = 0\}$$

$$O = \{o\dagger 1 \mid o \in O \wedge o\lceil 1 = 1\}$$

The last two conditions may be rephrased: if $o \in O$ then all of $o\dagger 1$, $0o$ and $1o$ are elements of O. This is probably a rather natural condition to assume and we shall investigate some consequences shortly. The nondeterministic semantics corresponding to using O as the set of oracles is

$$\underline{D}_O[\![cmd]\!] : S_\perp \to \mathcal{E}(S_\perp)$$

It is defined by

$$\underline{D}_O[\![cmd]\!] = \lambda s.\{s' \in S_\perp \mid \exists o,o' \in O: \underline{S}[\![cmd]\!](smash(s,o)) = smash(s',o')\}$$

but this is not a denotational definition in the sense of Scott and Strachey because the definition is not compositional. Clearly $\underline{D}_O$ is of the correct functionality when O is acceptable because then $\underline{S}[\![cmd]\!]$ specializes to the functionality $S_\perp * O_\perp \to S_\perp * O_\perp$. The set

$O = \{0,1\}^\omega$ used in the previous section is acceptable and, using the results in section 5.6, it is immediate to see that $\underline{D}_{\{0,1\}^\omega}$ equals $\underline{P}$.

The motivation behind assuming O to be acceptable is primarily to obtain the following two lemmas. A remark below shows that this cannot be achieved under weaker assumptions. The first lemma states that $\underline{P}$ and $\underline{D}_O$ agree if divergence (i.e. $\perp$ of $S_\perp$) is ignored.

Lemma 5.7:1. If O is acceptable then

$$\underline{D}_O[\![cmd]\!](s) - \{\perp\} = \underline{P}[\![cmd]\!](s) - \{\perp\}$$

holds for all commands cmd and $s \in S_\perp$. ///

Proof Since $\underline{P}$ is $\underline{D}_{\{0,1\}^\omega}$ the inclusion $\subseteq$ is immediate. The other inclusion follows from fact 5.6:1 and acceptability of O. For if $\perp \neq s' \in \underline{P}[\![cmd]\!](s)$ there is w such that $\underline{S}[\![cmd]\!](s,wo) = (s',o)$ for all $o \in \{0,1\}^\omega$ and hence some $o \in O$. But $wo \in O$ and therefore $s' \in \underline{D}_O[\![cmd]\!](s)$. ///

Acceptability also guarantees that "whenever one oracle is possible then all are" (as was considered in the proof of theorem 5.6:6).

Lemma 5.7:2. If O is acceptable then

$$\{smash(s',o') \mid s' \in \underline{D}_O[\![cmd]\!](s) \wedge o' \in O\} = \underline{C}[\![cmd]\!](s,O)$$

holds for all cmd and $s \in S_\perp$. ///

Proof Clearly $(\perp,\perp)$ is an element of the lefthand side iff it is an element of the righthand side. Next let $(s',o') \neq (\perp,\perp)$ be an element of the lefthand side. By fact 5.6:1 there is w such that $\underline{S}[\![cmd]\!](s,wo') = (s',o')$. Since $wo' \in O$ it follows that (s',o') is an element of the righthand side. Conversely let $(s',o') \neq (\perp,\perp)$

be an element of the righthand side. Then there is o∈O such that $\underline{S}[\![cmd]\!](s,o) = (s',o')$. By fact 5.6:1 and acceptability of O it follows that o'∈O. Hence (s',o') is an element of the lefthand side.

///

Remark Under reasonable assumptions it is necessary to assume that O is acceptable in order to obtain the previous lemma. First note that $O \neq \emptyset$ is required for $\underline{D}_O$ to be of the correct functionality.

Next assume that there are atomic actions x:=0 and x:=1 such that

$$\forall s \in S: \mathcal{A}[\![x:=0]\!](s) \neq \mathcal{A}[\![x:=1]\!](s)$$

Since S is non-empty we may choose s∈S. Suppose that O is chosen such that the equality in the previous lemma holds. Using it with s and

cmd = x:=0 or x:=1

shows the remaining conditions in the definition of acceptability.

///

Large parts of the development in section 5.6 can be carried through with any acceptable set O rather than only $O = \{0,1\}^\omega$. In particular theorem 5.6:3 ($\underline{C} = \hat{\underline{S}}$) and theorem 5.6:5 ($\underline{I} = \underline{P}$) will still hold. This implies that theorem 5.6:6 may be rephrased as

$$\underline{P}[\![cmd]\!] = \underline{D}_O[\![cmd]\!]$$

or equivalently as $\underline{D}_{\{0,1\}^\omega}[\![cmd]\!] = \underline{D}_O[\![cmd]\!]$. This equality may fail under reasonable assumptions about the expressiveness of atomic actions and boolean expressions. As an example take

cmd = x:=1 ; while x≠0 do (x:=x+1 or x:=x-1)

$O = \{w110110110\ldots \mid w \in \{0,1\}^*\}$

It follows from the previous lemma that the only difference possible

is that $\underline{D}_O[\![cmd]\!](s) = \underline{P}[\![cmd]\!](s) - \{\bot\}$, i.e. that $\underline{P}$ specifies divergence too often.

It is not clear what conditions to place on O such that $\underline{P}$ equals $\underline{D}_O$. The use of $O = \{0,1\}^\omega$ is not necessary however. To show this we shall assume that S and the sets of atomic actions and boolean expressions are all countable. Then the set of pairs (cmd,s) is countable and so is the set of pairs such that $\underline{P}[\![cmd]\!](s) \ni \bot$. Construct $X \subseteq \{0,1\}^\omega$ by including for each such (cmd,s) a string o such that $\underline{S}[\![cmd]\!](smash(s,o)) = (\bot,\bot)$. Clearly X is countable and therefore

$$O = \{w(o \dagger m) \mid w \in \{0,1\}^* \wedge o \in X\}$$

is countable and acceptable. Since O contains X it is immediate that $\underline{P}$ equals $\underline{D}_O$.

If "the choice of the path is done according to certain unknown parameters"/Egl75/ one might expect the choice to be made by some algorithm. In the present formulation this would imply that oracles should be "computable". The following lemma shows that if this view is taken then one cannot obtain $\underline{P}$. Define an oracle $o \in \{0,1\}^\omega$ to be recursive iff

$$\Theta(o) = \{n \mid \text{the n'th element of o is 1}\}$$

is a recursive set. An oracle is recursive iff there is a Turing Machine that lists all finite prefixes of o. We shall assume that the atomic actions and boolean expressions are sufficiently expressive that the language of commands may simulate arbitrary Turing Machines.

Lemma 5.7:3. If O is an acceptable set such that (for all cmd)

$$\underline{P}[\![cmd]\!] = \underline{D}_O[\![cmd]\!]$$

then O must contain the recursive oracles as a proper subset. ///

Proof We first show that all recursive oracles are needed. Let o be a recursive oracle. The concept of a Turing Machine with oracle X (a subset of the integers) is explained in /Rog67/ and a similar concept in /HoU179/. Consider the following informal description of such a Turing Machine:

```
n:=1 ; while n>0
    do compute the n'th element of o by simulating the Turing
        Machine for o;
    ask the oracle whether n is in it and let a be 1 if it is
        and 0 if not;
    if a differs from the n'th element then n:=0 else n:= n+1
```

By the assumptions about the expressiveness of the atomic actions and boolean expressions there is a program cmd and initial state $s \in S$ such that $\underline{S}[\![cmd]\!](s,o') = (\perp,\perp)$ iff the Turing Machine loops upon oracle $\Theta(o')$. For example the query of the oracle may be implemented by

```
a:=0 or a:=1
```

Clearly the Turing Machine loops iff it is supplied with the oracle $\Theta(o)$. Therefore $\underline{P}[\![cmd]\!](s) \ni \perp$ and

$$\underline{P}[\![cmd]\!](s) = \underline{D}_o[\![cmd]\!](s)$$

shows that $o \in O$.

We next show that some $o \in O$ must be non-recursive. Recall that there are recursively enumerable subsets X and Y of the integers such that $X \cap Y = \emptyset$ and such that no recursive set R satisfies $X \subseteq R$ and $R \cap Y = \emptyset$ /Rog67/. Let M_X and M_Y be Turing Machines recognizing X and Y and write $X \ni^k j$ and $Y \ni^k j$ for whether these accept j in at most

k steps. Let $\tau:N\to N\times N$ be a recursive enumeration of the pairs of integers such that $(i,j) = \tau(n)$ implies $i\leq n$ and $j\leq n$. Consider the following description of a Turing Machine M with oracle:

```
set up an empty list; n:=1;
while n>0
do  (j,k) := τ(n);
    ask the oracle whether it contains n and append 1 to the
      list if the answer is yes and otherwise append 0;
    if Xϑ^k j  and the j'th element on the list is 0
    or Yϑ^k j  and the j'th element on the list is 1
    then n:=0 else n:=n+1
```

Then M will loop upon oracle X but halt on any recursive oracle R. As before there is a program cmd and initial state s such that $\underline{S}[\![cmd]\!](s,o') = (\perp,\perp)$ iff M loops upon $\Theta(o')$. Since $\underline{P}[\![cmd]\!](s)\ni\perp$ it follows that O must contain some oracle that is not recursive.///

It has already been shown that there is an acceptable set O such that $\underline{D}_O$ differs from $\underline{P}$. The previous lemma makes it feasible to define a whole sequence of such examples. Let $(0^j1)^\omega$ mean the infinite string consisting of j 0's and a 1 etc. and define

$$O_i = \{w(0^j1)^\omega \mid w\in\{0,1\}^* \wedge j\leq i\}$$

Note that each O_i is acceptable and is a proper subset of both O_{i+1} and the set of recursive oracles. It follows that for all cmd and s:

$$\underline{D}_{O_1}[\![cmd]\!](s) \sqsubseteq \underline{D}_{O_2}[\![cmd]\!](s) \sqsubseteq \ldots$$

and using the proof of the previous lemma it follows that for all i there are cmd_i and s_i such that

$$\underline{D}_{O_i}[\![\mathrm{cmd}_i]\!](s_i) \neq \underline{D}_{O_{i+1}}[\![\mathrm{cmd}_i]\!](s_i)$$

Hence all $\underline{D}_{O_i}$ are different from one another. (Other examples of such a family $(O_i)_i$ may be obtained by defining O_i to be oracles of certain complexity classes.)

ANGELIC SEMANTICS

It follows from section 5.6 that the partly induced semantics $\underline{I}$ equals $\underline{P}$ even when $O = \{0,1\}^\omega$ is replaced by an arbitrary acceptable set O. This means that abstract interpretation (as developed in section 5.6) cannot be used to obtain a (compositional) denotational definition of $\underline{D}_O$ when it is not $\underline{P}$. In the remainder of this section we shall therefore study $\underline{D}_O$ for its own sake. A first question one may ask is whether a compositional definition can exist at all. Hopes for such a definition are motivated by the following lemma but none the less we shall see that the answer is no for some acceptable sets O.

Lemma 5.7:4. Let O be an acceptable set of oracles. Then

$$\underline{D}_O[\![\mathrm{act}]\!] = \{\} \cdot \mathcal{A}[\![\mathrm{act}]\!]$$

$$\underline{D}_O[\![\mathrm{cmd}_1;\mathrm{cmd}_2]\!] = \underline{D}_O[\![\mathrm{cmd}_2]\!] \,\square^P\, \underline{D}_O[\![\mathrm{cmd}_1]\!]$$

$$\underline{D}_O[\![\underline{\mathrm{if}}\ \mathrm{exp}\ \underline{\mathrm{then}}\ \mathrm{cmd}_1\ \underline{\mathrm{else}}\ \mathrm{cmd}_2]\!] = \mathrm{cond}^P_{\mathrm{exp}}(\underline{D}_O[\![\mathrm{cmd}_1]\!], \underline{D}_O[\![\mathrm{cmd}_2]\!])$$

$$\underline{D}_O[\![\mathrm{cmd}_1\ \underline{\mathrm{or}}\ \mathrm{cmd}_2]\!] = \underline{D}_O[\![\mathrm{cmd}_1]\!]\ \mathrm{or}^P\ \underline{D}_O[\![\mathrm{cmd}_2]\!]$$

$\underline{D}_O[\![\underline{\mathrm{while}}\ \mathrm{exp}\ \underline{\mathrm{do}}\ \mathrm{cmd}_1]\!]$ is a fixed point of

$$\lambda f.\ \mathrm{cond}^P_{\mathrm{exp}}(\ f \,\square^P\, \underline{D}_O[\![\mathrm{cmd}]\!], \{\}) \qquad ///$$

Proof The proof proceeds by structural induction.

Case cmd = act is straightforward.

Case cmd = $\mathrm{cmd}_1;\mathrm{cmd}_2$ follows from calculating

$$\underline{D}_O[\![\text{cmd}]\!](s) =$$

(because $\underline{S}[\![\text{cmd}_1]\!]$ specializes to $S_\perp * O_\perp \to S_\perp * O_\perp$)

$$\{s' \in S_\perp \mid \exists s'' \in S_\perp:\ \exists o,o',o'' \in O:$$

$$\text{smash}(s',o') = \underline{S}[\![\text{cmd}_2]\!](\text{smash}(s'',o'')) \wedge$$

$$\text{smash}(s'',o'') = \underline{S}[\![\text{cmd}_1]\!](\text{smash}(s,o))\} =$$

(by lemma 5.7:2)

$$\cup\{\underline{D}_O[\![\text{cmd}_2]\!](s'') \mid s'' \in \underline{D}_O[\![\text{cmd}_1]\!](s)\} =$$

$$(\underline{D}_O[\![\text{cmd}_2]\!] \ \square^P\ \underline{D}_O[\![\text{cmd}_1]\!])(s)$$

Case cmd = if exp then cmd_1 else cmd_2 is straightforward.

Case cmd = cmd_1 or cmd_2 is straightforward because O is acceptable.

Case cmd = while exp do cmd_1. The proof is easiest when assuming there is an atomic action skip such that $\mathcal{A}[\![\text{skip}]\!] = \lambda s.s$. Then reasoning as above shows that

$$\text{cond}^P_{\text{exp}}(\ f\ \square^P\ \underline{D}_O[\![\text{cmd}]\!], \{\}\ )$$

equals

$$\underline{D}_O[\![\text{if exp then (cmd;while exp do cmd) else skip}]\!]$$

when $f = \underline{D}_O[\![\text{while exp do cmd}]\!]$. But this equals f because $\underline{S}[\![\text{while} \ldots]\!]$ clearly equals $\underline{S}[\![\text{if} \ldots]\!]$. - If there is no atomic action skip the result is only slightly more cumbersome. ///

This lemma might be taken to mean that the compositional definition of $\underline{D}_O$ should be obtained from the equations defining P by replacing LFP by some other fixed point operator. It is not possible to do so for all acceptable choices of O. As an example define

exp = x

cmd_1 = x:=false or skip

cmd_2 = skip or x:=false

and

$$O = \{w000... \mid w \in \{0,1\}^*\}$$

It is immediate that $\underline{D}_O[\![cmd_1]\!]$ equals $\underline{D}_O[\![cmd_2]\!]$ and hence the two functionals (of which some fixed point is to be taken) are equal. So $\underline{D}_O[\![\underline{while}\ exp\ \underline{do}\ cmd_1]\!]$ would equal $\underline{D}_O[\![\underline{while}\ exp\ \underline{do}\ cmd_2]\!]$ and this is clearly wrong. Hence $\underline{D}_O$ cannot be defined along the lines suggested. ...

Remark It may well be argued that the example above is "unnatural" but it is not clear how to obtain any compositional definition of some $\underline{D}_O$ that is not $\underline{P}$. In particular, the approaches of /Par81/ and /deR81/ of defining a fair denotational semantics does not follow the approach outlined above. (One might take $\{0,1\}^\omega - \{000...,111...\}$ to be the set of "fair" oracles.) There $\underline{I}[\![cmd_1\ \underline{or}\ cmd_2]\!]$ essentially is $(\underline{I}[\![cmd_1]\!], \underline{I}[\![cmd_2]\!])$ and the analogue of the semantics for while describes, using least and greatest fixed points, a fair interleaving of $\underline{I}[\![cmd_1]\!]$ and $\underline{I}[\![cmd_2]\!]$. However, in this approach $\underline{I}[\![cmd_1\ \underline{or}\ cmd_2]\!]$ differs from $\underline{I}[\![cmd_2\ \underline{or}\ cmd_1]\!]$ and this is contrary to the intuition leading to P. ///

Since it is unclear how to define $\underline{D}_O$ compositionally we shall define "an upper bound" $\underline{A}$ along the lines suggested. The idea is that $\underline{P}$, $\underline{D}_O$ and $\underline{A}$ are equal when $\perp$ of $S_\perp$ is ignored and that if $\underline{A}$ specifies divergence then so does $\underline{D}_O$ and $\underline{P}$. The main task is to define a fixed point operator AFP. Define

$$AFP(F) = \bigsqcup\{f: S_\perp \to \mathcal{P}(S_\perp) \mid F(f)=f \wedge f \subseteq LFP(F)\}$$

for each monotonic and $\subseteq$-monotonic function F from $S_\perp \to \mathcal{P}(S_\perp)$ to itself. We shall see shortly that this definition makes sense. For

an informal motivation of the definition let

$$f = \underline{D}_O[\![\underline{\text{while}}\ \text{exp}\ \underline{\text{do}}\ \text{cmd}]\!]$$

$$F = \lambda f.\ \text{cond}^P_{\text{exp}}(\ f\ \square^P\ \underline{D}_O[\![\text{cmd}]\!]\ ,\{\})$$

It follows from the previous lemma that f is a fixed point of F and hence $\text{LFP}(F) \sqsubseteq f$. That $f \subseteq \text{LFP}(F)$ now means that LFP(F)(s) may specify divergence (i.e. contain $\perp$ of $S_\perp$) in situations where f(s) does not but otherwise they are equal. This corresponds to the difference between $\underline{P}$ and $\underline{D}_O$.

That AFP(F) is well-defined follows from:

Lemma 5.7:5. When F is monotonic and $\subseteq$-monotonic

$$\{f:S_\perp \to \mathcal{E}(S_\perp) \mid F(f)=f \wedge f \subseteq \text{LFP}(F)\}$$

is a complete lattice when ordered as in $S_\perp \to \mathcal{E}(S_\perp)$. The least element is LFP(F) and the greatest element is AFP(F). ///

Proof Let D be $\{f \mid F(f)=f \wedge f \subseteq \text{LFP}(F)\}$ ordered by $\sqsubseteq$ as in $S_\perp \to \mathcal{E}(S_\perp)$ and let E be $\{f \mid \text{LFP}(F) \sqsubseteq f \subseteq \text{LFP}(F)\}$ ordered similarly. Clearly D is the set of those elements of E that are fixed points of F. The assumptions about F guarantee that F specializes to a function from E to E. Throughout this proof $\bigsqcup$ refers to least upper bounds relative to $S_\perp \to \mathcal{E}(S_\perp)$ and $(\bigsqcup Y)(s)$ therefore is $\bigsqcup\{f(s) \mid f \in Y\}$. If Y is a nonempty subset of E then $\bigsqcup Y$ exists and is an element of E. Existence is because $f \in Y$ implies that f(s) is LFP(F)(s) or $\text{LFP}(F)(s)-\{\perp\}$ and these two elements are comparable. Concerning membership of E note that $\text{LFP}(F) \subseteq \bigsqcup Y$ is immediate and $\bigsqcup Y \subseteq \text{LFP}(F)$ is because both LFP(F)(s) and $\text{LFP}(F)(s)-\{\perp\}$ are subsets of LFP(F)(s).

Clearly LFP(F) is the least element of D. Let now Y be a subset of D. If Y is empty it is immediate that LFP(F) is the least upper

bound of Y in D. If Y is not empty $\bigsqcup Y$ exists and is an element of E but it is conceivable that it is not a fixed point of F. We therefore define by transfinite induction

$$F^{(\lambda)} = F(\bigsqcup(\{F^{(\kappa)} \mid \kappa<\lambda\} \cup Y))$$

It is a straightforward transfinite induction to show that each $F^{(\lambda)}$ exists and is an element of E (so that $\bigsqcup \ldots$ exists). It is straightforward to show that if $\kappa \leq \lambda$ then $F^{(\kappa)} \sqsubseteq F^{(\lambda)}$. As in section 5.1 this gives an ordinal λ such that $F^{(\lambda)}$ is a fixed point of F. Hence $F^{(\lambda)}$ is an element of D and it is clearly an upper bound of Y. If f is any other element of D that is an upper bound of Y it is straightforward to prove that $F^{(\lambda)} \sqsubseteq f$. Hence Y has a least upper bound in D. ///

It is possible to show that AFP(F) depends monotonically on F but this will not be needed. The following formula will be used in the proof of the theorem below.

Lemma 5.7:6. AFP(F) equals some $F_{(\lambda)}$ where

$$F_{(0)}(s) = \begin{cases} \{\bot\} & \text{if } LFP(F)(s) = \{\bot\} \\ LFP(F)(s) - \{\bot\} & \text{otherwise} \end{cases}$$

$$F_{(\lambda)} = F(\prod\{F_{(\kappa)} \mid \kappa<\lambda\}) \quad \text{for } \lambda>0 \qquad ///$$

Proof We shall use the terminology introduced in the proof of the previous lemma. Analogously to what was shown there it can be shown that every nonempty subset Y of E has greatest lower bound $\prod Y$ in E and that it is calculated pointwise. Since $F_{(0)}$ is an element of E it is a straightforward transfinite induction to show that each $F_{(\lambda)}$ is defined and is an element of E. Since $\kappa \leq \lambda$ implies $F_{(\kappa)} \sqsupseteq F_{(\lambda)}$ there is a λ such that $F_{(\lambda)}$ is a fixed point of F. Hence $F_{(\lambda)}$ is an

element of D and $F_{(\lambda)} \sqsubseteq AFP(F)$. That $AFP(F) \sqsubseteq F_{(\lambda)}$ follows by transfinite induction because $AFP(F) \sqsubseteq F_{(0)}$. ///

The semantics $\underline{A}$ is then defined by using AFP instead of LFP. In detail:

$$\underline{A}[\![cmd]\!] : S_\perp \to \mathcal{P}(S_\perp)$$

is defined by

$$\underline{A}[\![act]\!] = \{\} \cdot \mathcal{A}[\![act]\!]$$

$$\underline{A}[\![cmd_1 ; cmd_2]\!] = \underline{A}[\![cmd_2]\!] \square^P \underline{A}[\![cmd_1]\!]$$

$$\underline{A}[\![\underline{if}\ exp\ \underline{then}\ cmd_1\ \underline{else}\ cmd_2]\!] = cond^P_{exp}(\underline{A}[\![cmd_1]\!], \underline{A}[\![cmd_2]\!])$$

$$\underline{A}[\![cmd_1\ \underline{or}\ cmd_2]\!] = \underline{A}[\![cmd_1]\!]\ or^P\ \underline{A}[\![cmd_2]\!]$$

$$\underline{A}[\![\underline{while}\ exp\ \underline{do}\ cmd]\!] = AFP(\lambda f.\ cond^P_{exp}(f \square^P \underline{A}[\![cmd]\!], \{\}))$$

It is immediate to verify that $cond^P_{exp}$ and $\square^P$ are monotonic and $\subseteq$-monotonic. Therefore the functional of which AFP is taken satisfies the requirements. It is then easy to see that the equations define a strict function of functionality as stated.

It may not be very intuitive what $\underline{A}[\![cmd]\!]$ accomplishes. The following "operational characterisation" of $\underline{A}[\![cmd]\!]$ may therefore help. It is explained in more detail after the proof.

Theorem 5.7:7. For all commands cmd and states $s \in S_\perp$:

(1) $\underline{A}[\![cmd]\!](s) - \{\perp\} = \{s' \in S \mid \exists o, o' \in \{0,1\}^\omega : \underline{S}[\![cmd]\!](smash(s,o)) = (s', o')\}$

(2) $\underline{A}[\![cmd]\!](s) \ni \perp$ iff $\exists w \in \{0,1\}^* : \forall o \in \{0,1\}^\omega : \underline{S}[\![cmd]\!](smash(s,wo)) = (\perp,\perp)$

///

Proof The first result is an immediate consequence of the following result that is proved by structural induction on cmd:

$$\underline{P}[\![cmd]\!] \sqsubseteq \underline{A}[\![cmd]\!] \subseteq \underline{P}[\![cmd]\!]$$

Case cmd = act is immediate.

Case cmd = $cmd_1;cmd_2$ follows because $\square^P$ is monotonic and $\sqsubseteq$-monotonic.

Case cmd = if exp then cmd_1 else cmd_2 follows because $cond^P_{exp}$ is monotonic and $\sqsubseteq$-monotonic.

Case cmd = cmd_1 or cmd_2 follows because or^P is monotonic and $\sqsubseteq$-monotonic.

Case cmd = while exp do cmd_1. Abbreviate

$$F(f) = cond^P_{exp}(f \square^P \underline{P}[\![cmd_1]\!], \{\})$$
$$G(g) = cond^P_{exp}(g \square^P \underline{A}[\![cmd_1]\!], \{\})$$

It follows from the above that $f \sqsubseteq g$ implies $F(f) \sqsubseteq G(g)$ and that $g \sqsubseteq f$ implies $G(g) \sqsubseteq F(f)$. Since both $\sqsubseteq$ and $\subseteq$ are admissible it follows by the induction principle of section 5.1 that

$$LFP(F) \sqsubseteq LFP(G) \quad \text{and} \quad LFP(G) \subseteq LFP(F)$$

Since $LFP(G) \sqsubseteq AFP(G) \subseteq LFP(G)$ the result follows.

That the second result holds for all $s \in S_\perp$ is proved directly by structural induction.

Case cmd = act is immediate.

Case cmd = $cmd_1;cmd_2$. It is straightforward to prove that $\underline{A}[\![cmd]\!](s) \ni \perp$ holds iff $\underline{A}[\![cmd_1]\!](s) \ni \perp$ or (using the proof of the first part) $\underline{P}[\![cmd_1]\!](s) \ni s' \neq \perp$ and $\underline{A}[\![cmd_2]\!](s') \ni \perp$. Similarly $\underline{S}[\![cmd]\!](smash(s,wo)) = (\perp,\perp)$ holds iff $\underline{S}[\![cmd_1]\!](smash(s,wo)) = (\perp,\perp)$ or $\underline{S}[\![cmd_1]\!](s,wo) = (s',(wo)\dagger m)$ and $\underline{S}[\![cmd_2]\!](s',(wo)\dagger m) = (\perp,\perp)$. It is then straightforward to use the inductive hypotheses to show "only if" and "if".

Case cmd = if exp then cmd_1 else cmd_2 is straightforward (perform a case analysis upon $\mathcal{B}[\![exp]\!](s)$).

Case cmd = cmd_1 or cmd_2 is straightforward.

Case cmd = while exp do cmd_1. This proof is rather long and we shall consider "if" and "only if" seperately.

Proof of "if". The result is immediate for $s=\perp$ so assume $s\neq\perp$ below. Let Q(w,s) abbreviate

$$\forall o\in\{0,1\}^{\omega}: \underline{S}[\![\underline{\text{while}}\ \text{exp}\ \underline{\text{do}}\ \text{cmd}_1]\!](s,wo) = (\perp,\perp)$$

and define

$$P = \{(w,s) \mid Q(w,s) \wedge (Q(s',s) \Rightarrow |w| \leq |w'|)\}$$

It suffices to show that

$$(w,s)\in P \text{ implies } \underline{A}[\![\underline{\text{while}}\ \text{exp}\ \underline{\text{do}}\ \text{cmd}_1]\!](s) \ni \perp$$

We shall prove this result by induction on n where |w| = n. Note that Q(w,s) implies that $\mathcal{B}[\![\text{exp}]\!](s)$ is true and hence

$$\underline{A}[\![\underline{\text{while}}\ \ldots]\!](s) = \underline{A}[\![\underline{\text{while}}\ \ldots]\!]^{\dagger}(\underline{A}[\![\text{cmd}_1]\!](s))$$

$$\underline{S}[\![\underline{\text{while}}\ \ldots]\!](s,o) = \underline{S}[\![\underline{\text{while}}\ \ldots]\!](\underline{S}[\![\text{cmd}_1]\!](s,o))$$

When n=0 we have $\forall o$: $\underline{S}[\![\underline{\text{while}}\ \ldots]\!](s,o) = (\perp,\perp)$ and by (1) it follows that $\underline{A}[\![\underline{\text{while}}\ \ldots]\!](s) = \{\perp\}$ (as it cannot be empty). When n>0 we partition

$$P(n) = \{(w,s)\in P \mid |w| = n\}$$

into a disjoint union of three sets $P_1(n)$, $P_2(n)$, $P_3(n)$ and prove the result for each. For $P_1(n)$ take those elements (w,s) of P(n) such that $\forall o$: $\underline{S}[\![\text{cmd}_1]\!](s,wo) = (\perp,\perp)$. From the hypothesis of the structural induction it follows that $\underline{A}[\![\text{cmd}_1]\!](s) \ni \perp$ and hence $\underline{A}[\![\underline{\text{while}}\ \ldots]\!](s) \ni \perp$.

To define $P_2(n)$ and $P_3(n)$ we need some additional notation. For each (w,s) in $P(n)-P_1(n)$ there is $s' \in S$ and o,o' such that $\underline{S}[\![cmd_1]\!](s,wo) = (s',o') \neq (\perp,\perp)$. By fact 5.6:1 there exists $u = u_{w,s,s'}$ such that

one of u and w is a prefix of the other and

$\forall o: \underline{S}[\![cmd_1]\!](s,uo) = (s',o) \neq (\perp,\perp)$

For $P_2(n)$ take those (w,s) in $P(n)$ such that some $|u_{w,s,s'}|$ is greater than 0. For $P_3(n)$ take those (w,s) in $P(n)$ such that all $|u_{w,s,s'}|$ are 0. Clearly $P(n)$ is the disjoint union of $P_1(n)$, $P_2(n)$ and $P_3(n)$.

Consider $(w,s) \in P_2(n)$. Choose s' such that $|u_{w,s,s'}| > 0$ and write $u = u_{w,s,s'}$. By the definition of u it follows from (1) that $\underline{A}[\![cmd_1]\!](s) \ni s'$. Suppose $|u| \leq |w|$ and let w' be chosen such that $uw' = w$. Then

$\forall o: (\perp,\perp) = \underline{S}[\![\underline{\text{while}} \ldots]\!](s,wo) = \underline{S}[\![\underline{\text{while}} \ldots]\!](s',w'o)$

shows $Q(w',s')$. Since $|w'| < |w|$ it follows from the numerical induction that $\underline{A}[\![\underline{\text{while}} \ldots]\!](s') \ni \perp$ and this shows the result. Suppose next that $|u| > |w|$. Then

$\forall o: (\perp,\perp) = \underline{S}[\![\underline{\text{while}} \ldots]\!](s',o)$

and the result follows much as before.

To prove the result for elements of $P_3(n)$ we proceed as follows. Define $S(n) = \{s \mid \exists w: (w,s) \in P_3(n)\}$ and note that $\mathcal{B}[\![exp]\!](s)$ is true when $s \in S(n)$. Define a preorder upon the elements of S(n) by

$s \rightarrow s'$ iff $u_{w,s,s'}$ exists for some w (of length n)

We know that if such $u_{w,s,s'}$ exists it must have length 0. The

preorder is well-founded. For otherwise there is an infinite chain $s_0 \to s_1 \to s_2 \to \ldots$ and this would imply that

$$\forall o: \ \underline{S}[\![\underline{\text{while}} \ldots]\!](s_0,o) = (\perp,\perp)$$

contrary to $n>0$ and $(w,s_0) \in P_3(n) \subseteq P$ for some w. It therefore suffices to show that $\underline{A}[\![\underline{\text{while}} \ldots]\!](s) \ni \perp$ by induction on this well-founded order. So consider $s \in S(n)$ and suppose the result holds for all s' such that $s \to s'$ (or rather $s \to^+ s'$).

If there is some s' such that $s \to s'$ we have $\underline{A}[\![\underline{\text{while}} \ldots]\!](s') \ni \perp$ by the hypothesis of the well-founded induction. Also $\underline{A}[\![\text{cmd}_1]\!](s) \ni s'$ follows by (1) and this shows the result. Next suppose there is no s' such that $s \to s'$. By construction of $P_3(n)$ there is s' (but not in $S(n)$) and w (of length n) such that $u_{w,s,s'}$ exists. Since $|u_{w,s,s'}| = 0$ we have

$$\forall o: \ \underline{S}[\![\text{cmd}_1]\!](s,o) = (s',o)$$

$$\forall o: \ \underline{S}[\![\underline{\text{while}} \ldots]\!](s,o) = \underline{S}[\![\underline{\text{while}} \ldots]\!](s',o)$$

From the first equation and (1) it follows that $\underline{A}[\![\text{cmd}_1]\!](s) \ni s'$. Since $Q(w,s)$ the second equation shows $Q(w,s')$ and hence there is w' such that $(w',s') \in P$. Since $s' \notin S(n)$ and $|w'| \leq n$ the tuple (w',s') must have been considered previously in the numerical induction or when considering $P_1(n)$ or $P_2(n)$. Therefore $\underline{A}[\![\underline{\text{while}} \ldots]\!](s') \ni \perp$ and this shows the result.

Proof of "only if". The result is immediate for $s=\perp$ so assume $s \neq \perp$ below. Define $D = \bigcup_k D_k$, $D_0 = \{s\}$ and

$$D_{k+1} = \underline{A}[\![\text{cmd}_1]\!]^\dagger(\{s \in D_k \mid \mathcal{B}[\![\text{exp}]\!](s) \text{ is true}\})$$

If $s' \in D$ there is a minimal k such that $s' \in D_k$. Also there is a list $s=s_0, \ldots, s_k=s'$ with $s_i \in D_i$ and

$\mathcal{B}[\![\text{exp}]\!](s_i)$ is true when $i<k$

$\underline{A}[\![\text{cmd}_1]\!](s_i) \ni s_{i+1}$ when $i<k$

Since $s_i \neq \perp$ when $i<k$ it follows by (1) and fact 5.6:1 that there are w_i such that

$$\forall o: \underline{S}[\![\text{cmd}_1]\!](s_i, w_i o) = (s_{i+1}, o)$$

for $i+1<k$. This holds for $i = k-1$ as well if $s' \neq \perp$.

First suppose that $\perp \in D$. Let $s' = \perp$ and let s_i, w_i, k be as above. Since $s \neq \perp$ we know $k>0$. Then $\underline{A}[\![\text{cmd}_1]\!]s_{k-1} \ni \perp$ so by the hypothesis of the structural induction there is w_{k-1} such that

$$\forall o: \underline{S}[\![\text{cmd}_1]\!](s_{k-1}, w_{k-1} o) = (\perp, \perp)$$

Hence

$$\forall o: \underline{S}[\![\underline{\text{while}}\ \text{exp}\ \underline{\text{do}}\ \text{cmd}_1]\!](s_0, w_0 \ldots w_{k-1} o) = (\perp, \perp)$$

and this shows the result.

Next suppose $\perp \notin D$ but that there is $s' \in D$ with $\underline{A}[\![\underline{\text{while}}\ \ldots]\!](s') = \{\perp\}$. By (1) it is immediate that

$$\forall o: \underline{S}[\![\underline{\text{while}}\ldots]\!](s', o) = (\perp, \perp)$$

We have s_i, w_i and k as before and this gives

$$\forall o: \underline{S}[\![\underline{\text{while}}\ldots]\!](s, w_0 \ldots w_{k-1} o) = (\perp, \perp)$$

thereby showing the result.

Finally we show that one of the two previous cases must apply when $\underline{A}[\![\underline{\text{while}}\ldots]\!](s) \ni \perp$. For suppose

$$\forall s' \in D: \underline{A}[\![\underline{\text{while}}\ldots]\!](s') \neq \{\perp\}$$

Using the notation of lemma 5.7:6 we have $\forall s' \in D: F_{(0)}(s') \not\ni \perp$. We

show by transfinite induction that $\forall s' \in D: F_{(\lambda)}(s') \neq \perp$. By lemma 5.7:6 this then shows that the previous two cases are exhaustive. So let $\lambda > 0$ and $s' \in D$. If $\mathcal{B}[\![\text{exp}]\!](s')$ is false

$$F_{(\lambda)}(s') = \{s'\} \neq \perp$$

If $\mathcal{B}[\![\text{exp}]\!](s')$ is true then

$$\begin{aligned} F_{(\lambda)}(s') &= (\sqcap\{F_{(K)} \mid K<\lambda\})^{\dagger}(\underline{A}[\![\text{cmd}_1]\!](s')) \\ &\subseteq (\sqcap\{F_{(K)} \mid K<\lambda\})^{\dagger}(D) \\ &= \cup\{\cap\{F_{(K)}(s'') \mid K<\lambda\} \mid s'' \in D\} \end{aligned}$$

which does not contain $\perp$. Finally note that $\mathcal{B}[\![\text{exp}]\!](s')$ cannot be $\perp$ when $s' \in D$ as then $\underline{A}[\![\underline{\text{while}}\ldots]\!](s') = \{\perp\}$. ///

The first result stated in the theorem says that $\underline{A}[\![\text{cmd}]\!](s)$ and $\underline{P}[\![\text{cmd}]\!](s)$ are equal if one ignores divergence (i.e. $\perp$ of $S_{\perp}$). The second result gives the conditions under which $\underline{A}[\![\text{cmd}]\!](s)$ specifies divergence. This can be explained more intuitively as follows.

Imagine an operating system that executes cmd on s. Before starting execution a time bound is chosen by unbounded nondeterminism. Up to that time bound or is viewed as nondeterminism and some nondeterministic choice of path is made. After the time bound or is viewed as the amb operator mentioned in section 5.6. (This can be implemented by letting the operating system perform a breadth-first search for a terminating computation path.) So the program will terminate iff (after the time bound has lapsed) there is a sequence of "choices of path" such that it terminates. The semantics of cmd run under such a scheme is given by $\underline{A}[\![\text{cmd}]\!]$. In a sense $\underline{P}[\![\text{cmd}]\!]$ corresponds to the situation where the "time bound" is "infinity".

The difference between $\underline{A}$ and $\underline{P}$ is illustrated by the following examples.

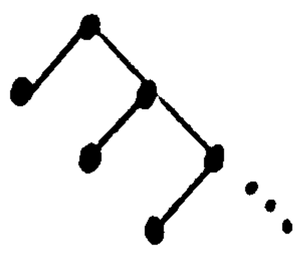
Example Consider the programs

$$cmd_1 = x:=true \ ; \ \underline{while}\ x\ \underline{do}\ (x:=false\ \underline{or}\ \underline{skip})$$

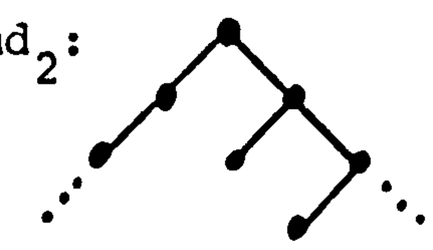
$$cmd_2 = (\underline{while}\ true\ \underline{do}\ \underline{skip})\ \underline{or}\ cmd_1$$

The "operational semantics" of these programs are best illustrated by the following execution trees:

cmd_1:



cmd_2:



Since both programs $\underline{may}$ loop we have

$$\underline{P}[\![cmd_1]\!] = \underline{P}[\![cmd_2]\!] = \lambda s.\ \{\perp, s[false/x]\}$$

Further cmd_1 may always be brought to completion so that

$$\underline{A}[\![cmd_1]\!] = \lambda s.\ \{s[false/x]\}$$

This differs from $\underline{A}[\![cmd_2]\!] = \underline{P}[\![cmd_2]\!]$ because in cmd_2 a path $\underline{may}$ be chosen such that thereafter the program $\underline{has}\ \underline{to}$ loop. ///

Example Consider the programs

cmd_3 = x:=true ; y:=0 ; $\underline{while}$ x
$\underline{do}$ (y:=y+1 $\underline{or}$ $\underline{if}$ y≤27 $\underline{then}$ x:=false $\underline{else}$ $\underline{skip}$)

cmd_4 = x:=true ; y:=0 ; $\underline{while}$ x
$\underline{do}$ (y:=y+1 $\underline{or}$ x:=false)

The semantics of cmd_3 is the same under $\underline{P}$ and $\underline{A}$ and is

$$\lambda s.\ \{\perp\} \cup \{s[false/x, m/y] \mid 0 \leq m \leq 27\}$$

The semantics of cmd_4 is different under $\underline{P}$ and $\underline{A}$:

$$\underline{P}[\![cmd_4]\!] = \lambda s.\{\perp\} \cup \{s[false/x,m/y] \mid 0 \leq m\}$$

$$\underline{A}[\![cmd_4]\!] = \lambda s.\ \{s[false/x,m/y] \mid 0 \leq m\}$$

Note that $\underline{A}[\![cmd_4]\!]$ specifies unbounded nondeterminism. ///

It follows from the theorem and lemma 5.7:1 that all of $\underline{P}[\![cmd]\!]$, $\underline{D}_O[\![cmd]\!]$ (for O acceptable) and $\underline{A}[\![cmd]\!]$ are equal when divergence (i.e. $\perp$ of $S_\perp$) is ignored. It is clear that if $\underline{A}[\![cmd]\!](s)$ specifies divergence then so must $\underline{D}_O[\![cmd]\!](s)$, and if $\underline{D}_O[\![cmd]\!](s)$ specifies divergence then so must $\underline{P}[\![cmd]\!](s)$. This shows that

$$\underline{P}[\![cmd]\!] \subseteq \underline{D}_O[\![cmd]\!] \subseteq \underline{A}[\![cmd]\!]$$

$$\underline{A}[\![cmd]\!] \subseteq \underline{D}_O[\![cmd]\!] \subseteq \underline{P}[\![cmd]\!]$$

holds for all acceptable sets O. It further follows that $\underline{P}$ can be used to show absence of divergence (wrt. $\underline{D}_O$) and $\underline{A}$ to prove presence of divergence.

6 CONCLUSIONS AND FURTHER WORK

To assess what has been achieved in this thesis it is appropriate first to recall why data flow analysis (including abstract interpretation) is worth doing. This was explained in section 2.1 and may be summarised as: To associate information with points in the program so as to show the safeness of performing program transformations that in general are not meaning preserving.

Viewing the development in these broad terms there are a number of major issues that have not been addressed.

- Program points should be introduced into the semantics in such a way that an approximating interpretation specifies the information possible at each program point. The development in this thesis may be viewed as working with only one program point: the end of the program. For ideas in this direction see /Nie82/.
- Program transformations should be formulated and proven safe to apply when the information specified above satisfies certain criteria. For ideas about how this might be done consult /Nie81b/.
- One should consider ways of implementing the data flow analyses described by approximating interpretations. This will be considered further below.

These issues are in a sense orthogonal to the aims of this thesis. (It is to be hoped that they are sufficiently orthogonal that they can be added without too much trouble.) The aims of the thesis were primarily to overcome the limited applicability of previous formulations of abstract interpretation: either they were based on viewing programs as flowcharts, which is hardly a general model, or else a mere study of toy languages. Turning to an assessment based on these aims it is convenient to consider chapters 2 to 4 seperately from chapter 5.

Chapters 2 to 4 considered how to extend abstract interpretation to deal with a larger class of languages. This was done by basing the development upon a metalanguage for denotational semantics. The main achievements of this development are:

- The notion of two levels of types allows for a precise description of the method, i.e. of where to introduce power-domains in the collecting semantics.
- The metalanguage gives a reasonable degree of generality.
- The tensor product allows for a generalisation of the "relational method" for cartesian product and smash product.
- A framework is given for how to change between various data flow analysis methods such as "relational" and "independent attribute" methods (section 4.3).
- The notion of "inducing" one data flow analysis from another has been generalised and "expected definitions" have been studied. This should make it more convenient to describe a particular data flow analysis as an approximating interpretation.

There are several possibilities for extension of the development. The most "urgent" extensions are:

- To allow "storable procedures", i.e. to extend the type structure with gt::=ft. This remedies the major limitation of the metalanguage and is discussed further below.
- To allow Y not always to be LFP. Some uses of this will be mentioned below.
- Analogously to allow rec not always to mean REC (i.e. not always to specify the initial solution).

Some further possibilities are:

- Introduce notation for expressions of type gt. This is further considered below.
- All the analyses are "forward" in the sense of propagating information from left to right. Dually one may consider how to handle "backward" analyses (see e.g. /Cou81/).

Even more remote goals are to extend the metalanguage with polymorphism, abstract data types, concurrency etc.

It is only sensible to anticipate that further research may cast doubt upon some of the decisions made in the development so far. It may be that the bottom-level metalanguage is not sufficiently close to operational intuitions. In particular it may be that $\underline{\times}$ should be discarded and $\underline{+}$ used instead. If also $\underline{\perp}$ is removed it would be feasible to require (in section 4.4) that all irreducible elements are either $\perp$ or an atom. Some potentially desirable consequences of doing so were outlined in the final remark in section 4.4.

Chapter 5 was concerned with bringing aspects of termination within the world of abstract interpretation. Some achievements of this are:

- More data flow analyses can be handled (e.g. section 5.5).
- Nondeterminism has formally been shown to be a data flow analysis problem. This is mainly of conceptual interest.
- Focus has been placed on the different partial orders $\sqsubseteq$ and $\leq$. This gives better insight into the essence of abstract interpretation and clarifies some intuitive remarks in /Nie82/. (A first attempt at formalising this appeared already in /Myc81/.)

Many extensions need to be made before the development of chapter 5 can be made more general than just studying toy languages. Some possibilities are mentioned below and they seem to require "hard work":

- The powerdomain should be made more generally applicable (there are too few benign fdcpo's).
- The development should be performed for a metalanguage (as in chapters 2 to 4).
- The idea behind partly collecting and partly inducing is that only some of the arguments are viewed as describing sets of values. It would be natural to allow for gradually viewing more and more arguments this way, e.g. to obtain data flow analysis for a nondeterministic language. We shall return to this below.

One question that may be asked is whether the use of powerdomains, in particular $\mathcal{E}(\ldots)$, is inherently satisfactory. This will be dis-

cussed in some detail below and it seems that the answer is no.

The remainder of this chapter considers in more detail some of the directions for further research listed above. They are grouped under the headings "implementation issues", "extending the power of the metalanguage" (for chapters 2 to 4) and "the strong approach" (for chapter 5).

Implementation issues

There are at least two views of how to implement a data flow analysis. One is that there should be a system that directly executes the approximating semantics. This seems to be the view taken in /Don81/. Another is that the approximating semantics is merely a mathematical formulation of the (approximate) effect of computing data flow information by traditional means, e.g. as MFP or MOP solutions to certain systems of equations. This is the view taken in /Nie82/. In the discussion below we shall ignore that the equations can hardly be formulated if program points have not been introduced.

So let $\underline{I}$ be an approximating semantics, e an expression of type $gt \underline{\rightarrow} gt$, env an (empty) environment and l and element of $\underline{I}[\![gt]\!]$. The desired evaluation system accepts $\underline{I}$, e, env and l and calculates $\underline{I}[\![e]\!](env)(l)$. There is a similar situation for the standard semantics and here the SIS system of /Mos79/ can be used to evaluate $S[\![e]\!](env)(d)$. The SIS system cannot be used to compute $\underline{I}[\![e]\!](env)(l)$. Some "minor" reasons are that SIS does not contain powerdomains and does not contain a (binary) least upper bound operation. The latter means that the expected definition of $\underline{I}(cond)$ cannot be defined.

A deeper reason has to do with the way the fixed point operator Y is treated. Essentially SIS is a rewrite system for the typed λ-calculus and Y is handled by rewriting

$$(Y(\lambda x.e))\ldots \text{ to } (e[Y(\lambda x.e)/x])\ldots$$

For a typical example let

$$e = \text{cond}(e_0, x \blacksquare e_1, e_2)$$

(corresponding to the semantics of a while-loop) where neither e_0, e_1 nor e_2 contains x. The unfolding continues as long as e_0 evaluates to true and Y disappears altogether once e_0 evaluates to false. Should e_0 always evaluate to true the system loops, but one can hardly expect otherwise as the semantics then is ⊥.

This strategy is not valid for evaluating $\underline{I}[\![e]\!](env)(1)$ and therefore another strategy must be developed. For generally some parts of the argument will always be passed to $x \blacksquare e_1$ and other parts to e_2. Therefore Y need never disappear and the system will loop. (This is indeed what happens in /Don81/ as was said in chapter 1.) On the other hand if $\underline{I}[\![gt]\!]$ is finite it will be the case that $\underline{I}[\![e]\!](env)(1)$ is obtained after a finite number of unfoldings (when all remaining Y are viewed as ⊥). It is therefore not acceptable that the system loops and it would be interesting to develop a suitable "rewrite rule".

The other approach was to assume that data flow analyses are computed as MOP and MFP solutions to certain equations. We shall write MOP(1) and MFP(1) for the information calculated at the end of the program given it is 1 at the beginning. It is known that $MOP(1) \sqsubseteq MFP(1)$ and the author conjectures that it extends to

$$MOP(1) \sqsubseteq \underline{I}[\![e]\!](env)(1) \sqsubseteq MFP(1)$$

and that neither inequality is an equality in general. Some intuitive arguments are presented below.

The intuition for why $\underline{I}[\![e]\!](env)(l)$ is between MOP(l) and MFP(l) but equal to neither is that its definition may be viewed as a mixture of MOP and MFP ingredients. An MFP ingredient may be illustrated for $e = f_1 \sqcap cond(\ldots, f_2, f_3)$. Here

$$\underline{I}[\![e]\!](env)(l) = f_1(f_2(l_{tt}) \sqcup f_3(l_{ff})) = MFP(l)$$

$$MOP(l) = f_1(f_2(l_{tt})) \sqcup f_1(f_3(l_{ff}))$$

which may differ when f_1 is not additive. A MOP ingredient may be illustrated for $e = Y(\lambda x.\lambda y.e')$. Here

$$\underline{I}[\![e]\!](env) = LFP(\lambda f.\lambda l'.\ \underline{I}[\![e']\!](env[f/x, l'/y]))$$

which is close to MOP whereas MFP essentially is

$$\lambda l.\ LFP(\lambda f.\lambda l'.\ \underline{I}[\![e']\!](env[f/x, (l \sqcup l')/y])) \qquad (*)$$

(The latter claim is based on /Nie82 p.281/.) When $\underline{I}[\![\lambda y.e']\!](env)$ is not additive the MFP and MOP solutions may well differ.

In data flow analysis it is commonly agreed that the MOP solution is more accurate than the MFP solution. This means that

$$abs \circ \underline{C}[\![e]\!](env) \circ con \sqsubseteq MOP \sqsubseteq MFP$$

However, it is usually the MFP solution that is computed in practice. For the MFP solution is decidable if all $\underline{I}[\![gt]\!]$ are of finite height (though possibly infinite) whereas the MOP solution is undecidable in some such cases /KaUl77/. The author conjectures that $\underline{I}[\![e]\!](env)$ behaves as MOP. One may therefore consider how to make $\underline{I}[\![e]\!](env)$ equal to MFP. One possibility is to enforce additivity of functions as then the MOP and MFP solutions agree. Another possibility is to redefine the interpretation of $Y(\lambda x.\lambda y.e')$ so it becomes the formula

(*) given above. Redefinition of Y is considered further below.

Extending the power of the metalanguage

There are other movivations for varying the interpretation of I⟦Y...⟧ than merely to obtain the MFP solution. One might desire even more approximative solutions than the MFP solution, e.g. if not all I⟦gt⟧ are of finite height or if it "takes too long" to compute the MFP solution. This is the motivation behind the "widening" and "narrowing" ideas of /CoCo77b/ and it is be hoped that these ideas can be incorporated by suitably interpreting Y. The interpretation of Y can be allowed to vary simply by letting an interpretation specify the operator to be used. The expected definition (in the terminology of section 4.4) then is LFP. The predicate $\leq_{con,(t\to t)\to t}$ will be useful for relating various interpretations of Y. One can extend "inducing" to Y of type $(t\to t)\to t$ by using the suggestion in section 4.2. When t = ft this gives

$$abs\cdot Y(\lambda g.\ con\cdot G(abs\cdot g\cdot con)\cdot abs)\cdot con$$

as the induced version of Y applied to G.

Quite analogously one may consider varying the interpretation of rec so that it is not always REC (giving the initial solution). Taking a recursive domain

recX. N*X+O

as an example one might hope to obtain a finite domain of descriptions of such lists. This may be the way to incorporate the ideas in /Jon81b/ about finding "safe approximate descriptions of computations by algorithms which manipulate recursive data structures".

A major limitation of the metalanguage is that "storable procedures" cannot be accomodated, i.e. that the type structure does not allow gt::=ft. To overcome this it will probably be necessary to find a locally continuous semi-functor $\ominus$ over $\underline{\underline{\text{ACLs}}}$ such that

$$\wp(A) \ominus \wp(B) = \wp(A \to B)$$

$$\wp(f) \ominus \wp(g) = \wp(\lambda h.\ g \circ h \circ f)$$

(ignoring that $\ominus$ will be contravariant in one argument). This amounts to defining a relational method for the bottom-level domain constructor $\underline{\to}$ and make it work not only for powerdomains. An independent attribute method might be function space $\to$ as has been used throughout this thesis. One application of a relational method for $\underline{\to}$ might be to extend abstract interpretation to handle properties like commutativity and associativity of functions. Such properties are frequently used when transforming applicative programs. Another application might be to express that the result of some function is always less than its argument. This may be the way to overcome some of the limitations of showing termination that were discussed at the end of section 5.5.

This situation is similar to the one with the tensor product. So one might try to construct $A \ominus B$ by completing a quasi order obtained from a logic. Terms might be given by

$$t ::= \ominus(a,b) \mid t \sqcup t' \mid t \sqcap t'$$

where $\ominus(a,b)$ is to be thought of as $\lambda a'.\ a' \sqsupseteq a \to b, \bot$. Some axioms might be

$$\ominus(a \sqcup a', b) = \ominus(a,b) \sqcap \ominus(a',b)$$

$$\ominus(a, b \sqcup b') = \ominus(a,b) \sqcup \ominus(a,b')$$

But even for the tensor product it was difficult to argue that it was the "right" generalisation from powerdomains to complete lattices and it is unlikely to be any easier here.

The metalanguage (section 2.1) has notation for expressions of type gt→gt but not for expressions of type gt. It might be convenient to introduce expressions of type gt by incorporating

$$\underline{\lambda}x.e \quad \text{and} \quad e\underline{(}e'\underline{)}$$

As long as gt::=ft is not allowed there will necessarily be some restrictions upon the use of $\underline{\lambda}$-abstraction. The meaning of these constructs is clear in the standard semantics:

$$\underline{S}[\![\underline{\lambda}x.e]\!](env) = \lambda v.\underline{S}[\![e]\!](env[v/x])$$

$$\underline{S}[\![e\underline{(}e'\underline{)}]\!](env) = (\underline{S}[\![e]\!](env))(\underline{S}[\![e']\!](env))$$

In an approximating interpretation $\underline{I}$ it may be better to use

$$\underline{I}[\![\underline{\lambda}x.e]\!](env) = lin(\lambda w.\ \underline{I}[\![e]\!](env[w/x]))$$

$$\underline{I}[\![e\underline{(}e'\underline{)}]\!](env) = (\underline{I}[\![e]\!](env))(\underline{I}[\![e']\!](env))$$

The use of lin may be important when x occurs more than once in e. As an example let e be x*x+1 and view $\underline{I}[\![\underline{\lambda}x.e]\!](env)$ as a function from $V = \{\bot,-,0,+,\top\}$ to itself. When applied to $\top$ one expects to get + rather than $\top$.

A consequence of the above suggestion is to cast some doubt upon the expected definition of composition in the bottom-level metalanguage. (It was defined in section 4.4 as f□g = f°g.) Intuitively one would expect f□g and $\underline{\lambda}x.f\underline{(}g\underline{(}x\underline{)}\underline{)}$ to denote the same. This holds in the standard semantics but in an approximating semantics the expected definitions give

$\underline{I}[\![f \circ g]\!](env) = f^{\cdot}g$

$\underline{I}[\![\underline{\lambda}x.f\underline{(}g\underline{(}x\underline{)}\underline{)}]\!](env) = lin(f^{\cdot}g)$

where $f = \underline{I}(f)$ and $g = \underline{I}(g)$. These may differ even when $f = lin(f)$ and $g = lin(g)$ as was shown in section 4.1. So one might consider to let $lin(f^{\cdot}g)$ be the expected definition of $f \circ g$.

The strong approach

The idea behind the partly collecting semantics was to pass from $f:D \times E \to F$ in the standard semantics to $g:D \times \mathcal{P}(E) \to \mathcal{P}(F)$. This may be approximated by an approximating semantics $h:D \times B \to C$. At this point it is conceivable that one may want to extend the data flow analysis considerations to include also the first argument. So from g one would pass to $g':\mathcal{P}(D \times E) \to \mathcal{P}(F)$ and this may be approximated by using approximations to $\mathcal{P}(D \times E)$ and $\mathcal{P}(F)$. From h it is less clear what to pass to. Perhaps it should be $h':\mathcal{P}(D) \star B \to C$ for $\star$ some kind of tensor product. Again this may be approximated by using approximations to $\mathcal{P}(D)$. It is not clear how to accomodate such a development. A potential application is to perform data flow analysis for a nondeterministic language.

To agree with data flow analysis intuitions it should be possible to view the powerdomain as a certain collection of subsets. For this reason it may be well not to base the development on the powerdomains of /Plo82/: for it was conjectured in section 5.2 that these were not atomically generated. However, it may be that the convexity assumption used in the definition of $\mathcal{P}(D)$ excludes subsets of interest. The following example seems to show this.

Example Consider an imperative language with procedures of one argument and let S and V be sets of states and values respectively. A procedure may be modelled as a function from $S_\perp \times V_\perp$ to $S_\perp$ and evaluation of an argument as a function from $S_\perp$ to $S_\perp \times V_\perp$. Consider the following program:

```
let p(value y) be skip in
    let q(name y) be skip in
    if x is even then LOOP else skip;
    call p(x)
```

The set of states possible before call p(x) will include $\perp$ and some $s \in S$. Evaluation of x in these states give $\perp$ and some $v \in V$, respectively. So the set of state and value pairs possible before transfer to p includes $(\perp,\perp)$ and (s,v). By convexity it must include $(s,\perp)$ as well. This is unfortunate because p and q differ upon $(s,\perp)$ although they are equal upon $(\perp,\perp)$ and (s,v). It is safe to replace the call to p by a call to q but the collecting semantics will produce a set from which this cannot be seen! ///

It is unclear how to get around this problem. If convexity is abandoned then $\mathcal{R}(D)$ is not even partially ordered by the Egli-Milner order. If a powerdomain like $\mathcal{E}(D)$ is to be used it seems that the collecting semantics is useless for some program transformations. Neither outlook is comforting.

REFERENCES

/AhU178/ A.V.Aho, J.D.Ullman: Principles of Compiler Design, Addison-Wesley, 1978.

/ApP182/ K.Apt, G.D.Plotkin: Countable Nondeterminism and Random Assignment, Dept. of Comp. Science Report CSR-98-82, University of Edinburgh, 1982.

/ArMa75/ M.A.Arbib, E.G.Manes: Arrows, Structures and Functors: The Categorical Imperative, Academic Press, 1975.

/Ban80/ H-J.Bandelt: The tensor product of continuous lattices, Mathematische Zeitschrift, vol 172 (1980) pp. 89-96.

/BaSa74/ A.M.Bauer, H.J.Saal: Does APL really need run-time checking? Software-Practice and Experience, vol.4 (1974), pp. 129-138.

/CoCo76/ P.Cousot, R.Cousot: Static determination of dynamic properties of programs, in: Proceedings of the 2nd international symposium on programming, Paris, 1976.

/CoCo77a/ P.Cousot, R.Cousot: Static determination of dynamic properties of generalized type unions, Sigplan Notices, vol 12 no 3 (1977), pp. 77-94.

/CoCo77b/ P.Cousot, R.Cousot: Abstract Interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints, in: Conf. Record of the 4th ACM Symposium on Principles of Programming Languages, 1977.

/CoCo78/ P.Cousot, R.Cousot: Static determination of dynamic properties of recursive procedures, in: Formal Descriptions of Programming Concepts, E.J.Neuhold (ed.), North-Holland Publishing Company, 1978.

/CoCo79/ P.Cousot, R.Cousot: Systematic design of program analysis frameworks, in: Conf. Record of the 6th ACM Symposium on Principles of Programming Languages, 1979.

/Cou81/ P.Cousot: Semantic foundations of program analysis, in: Program Flow Analysis: Theory and Applications, S.S.Muchnick, N.D.Jones (eds.), Prentice-Hall 1981.

/Don78/ V.Donzeau-Gouge: Utilisation de la sémantique dénotationelle pour l'étude d'interprétations non-standard, IRIA report no. 273, France, 1978.

/Don81/ V.Donzeau-Gouge: Denotational definition of properties of program computations, in: Program Flow Analysis: Theory and Applications, S.S.Muchnick, N.D.Jones (eds.), Prentice-Hall, 1981.

/Egl75/ H.Egli: A Mathematical Model for Non-deterministic Computations, Forschungsinstitut für Mathematik, ETH Zürich, 1975.

/GHKLMS80/ G.Gierts, K.H.Hofmann, K.Keimel, J.D.Lawson, M.Mislove, D.J.Scott: A Compendium of Continuous Lattices, Springer-Verlag, Berlin, Heidelberg, New York, 1980.

/Gor79/ M.J.C.Gordon: The Denotational Description of Programming Languages: An Introduction, Springer-Verlag, 1979.

/GrWe76/ S.L.Graham, M.Wegman: A fast and usually linear algorithm for global flow analysis, JACM, vol. 23 no.1 (1976), pp. 172-202.

/Hal60/ P.R.Halmos: Naive Set Theory, D. van Nostrand Company, 1960.

/Ham82/ A.G.Hamilton: Numbers, Sets and Axioms: The apparatus of Mathematics, Cambridge University Press, 1982.

/Hec77/ M.S.Hecht: Flow Analysis of Computer Programs, North-Holland, 1977.

/Hen82/ M.Hennessy: Powerdomains and Nondeterministic Recursive Definitions, in: 5th Int. Symp. on Programming, Lecture Notes in Computer Science 137, Springer-Verlag, 1982.

/HoUl79/ J.Hopcroft, J.D.Ullman: Introduction to Automata Theory, Languages, and Computation, Addison-Wesley, 1979.

/Joh82/ P.T.Johnstone: Stone Spaces, Cambridge studies in advanced mathematics 3, Cambridge University Press, 1982.

/Jon81a/ N.D.Jones, S.S.Muchnick: Complexity of flow analysis, inductive assertion synthesis and a language due to Dijkstra, in: Program Flow Analysis: Theory and Applications, S.S.Muchnick, N.D.Jones (eds.), Prentice-Hall, 1981.

/HeAs80/ M.C.B. Hennessy, E.A.Ashcroft: A Mathematical Semantics for a Nondeterministic Typed λ-Calculus, Theoretical Computer Science 11 (1980) 227-245.

/Jon81b/ N.D.Jones: Flow Analysis of Lambda Expressions, in: Proceedings ICALP 1981, Lecture Notes in Computer Science 115, Springer-Verlag, 1981, pp. 114-128.

/KaUl77/ J.B.Kam, J.D.Ullman: Monotone data flow analysis frameworks, Acta Informatica, vol 7 (1977), pp. 305-317.

/Kil73/ G.A.Kildall: A unified approach to global program optimization, in: Conf. Record of ACM Symposium on Principles of Programming Languages, 1973.

/LeSm81/ D.J.Lehman, M.B.Smyth: Algebraic specifications of data types: ä synthetic approach, Mathematical Systems Theory, vol 14 (1981), pp. 97-139.

/Mac71/ S.MacLane: Categories for the Working Mathematician, Springer-Verlag, 1971.

/Mar76/ G.Markowsky: Chain-complete posets and directed sets with applications, Algebra Universalis 6 (1976), pp. 53-68.

/McC67/ J.McCarthy: A basis for a mathematical theory of computation, in: P.Braffort, D.Hirschberg (eds.), Computer Programming and Formal Systems, North-Holland, 1967, pp. 33-70.

/Men63/ E.Mendelson: Introduction to Mathematical Logic, D. van Nostrand Company, Inc., 1963.

/Mil75/ R.Milner: Processes: A Mathematical Model of Computing Agents, in: H.Rose, J.Shepherdson (eds.), Logic Colloquium '73, North-Holland, Amsterdam, 1975, pp. 157-174.

/MiSt76/ R.Milne, C.Strachey: A Theory of Programming Language Semantics, Chapman and Hall, London, 1976.

/Mos79/ P.D.Mosses: SIS - Semantics Implementation System: Reference Manual and User Guide, DAIMI report no. MD-30, Denmark, 1979.

/Myc80/ A.Mycroft: The theory and practice of transforming call-by-need into call-by-value, in: Proc. 4th Int. Symposium on Programming, Lecture Notes in Computer Science 83, Springer-Verlag, 1980, pp. 269-281.

/Myc81/ A.Mycroft: Abstract Interpretation and Optimising Transformations for Applicative Programs, Ph.D.-thesis, University of Edinburgh, Scotland, 1981.

/MyNi83/ A.Mycroft, F.Nielson: Strong abstract interpretation using power domains, in: Proceedings ICALP 1983, Lecture Notes in Computer Science 154, Springer-Verlag 1983, pp. 536-547.

/Nie81a/ F.Nielson: Semantic Foundations of Data Flow Analysis, M.Sc.-thesis, University of Aarhus, Denmark, 1980. Also as DAIMI report PB-131, Denmark 1981.

/Nie81b/ F.Nielson: Program transformations in a denotational setting, DAIMI report PB-140, Denmark 1981.

/Nie82/ F.Nielson: A denotational framework for data flow analysis, Acta Informatica, vol. 18 (1982), pp. 265-287.

/Nie83/ F.Nielson: Towards viewing nondeterminism as abstract interpretation, in: Proceedings 3rd Conference on Foundations of Software Technology and Theoretical Computer Science, Bangalore, India, 1983.

/Par81/ D.Park: A predicate transformer for weak fair interation, in: Proc. 6th IBM Symposium on Mathematical Foundations of Computer Science, Hakone, Japan, 1981.

/Ple81/ U.F.Pleban: Preexecution Analysis based on Denotational Semantics, Ph.D.-thesis, University of Kansas, 1981.

/Plo76/ G.D.Plotkin: A powerdomain construction, SIAM J. Comput., vol 5 no 3 (1976), pp. 452-487.

/Plo82/ G.D.Plotkin: A Powerdomain for Countable Nondeterminism, in: Proc. ICALP 82, Lecture Notes in Computer Science 140, Springer-Verlag, Berlin, 1982, pp. 418-428.

/PloLN/ G.D.Plotkin: Lecture Notes (chapters 1 to 8); chapters 1 to 5 presented in Pisa, Italy, 1978.

/PloPS/ G.D.Plotkin: personal communication.

/Rey74/ J.C.Reynolds: On the Relation between Direct and Continuation Semantics, in: Proceedings 2nd ICALP 1974, Lecture Notes in Computer Science 14, pp. 141-156.

/deR81/ W.deRoever: A formalism for reasoning about fair termination, in: Logics of Programs, Lecture Notes in Computer Science 131, Springer-Verlag, Berlin, pp. 113-121.

/Rog67/ H.Rogers: Theory of Recursive Functions and Effective Computability, McGraw-Hill, 1967.

/Ros77/ B.K.Rosen: High-level data flow analysis, CACM vol 20 no 10 (1977), pp. 712-724.

/Sco76/ D.S.Scott: Data types as lattices, SIAM Journal on Computing, vol 5 (1976), pp. 522-587.

/Sco82/ D.S.Scott: Domains for denotational semantics, in: Proc. ICAPL 1982, Lecture Notes in Computer Science 140, Springer-Verlag, 1982, pp. 577-613.

/Smy78/ M.B.Smyth: Powerdomains, J. Comput. System Sci., vol 16 (1978), pp. 23-36.

/SmPl82/ M.B.Smyth, G.D.Plotkin: The category-theoretic solution of recursive domain equations, SIAM J. Comput., vol 11 no 4 (1982), pp. 761-783.

/Sto77/ J.E.Stoy: Denotational Semantics: The Scott-Strachey Approach to Programming Language Theory, MIT Press, 1977.

/Tar55/ A.Tarski: A lattice-theoretical fixed point theorem and its applications, Pacific Journal of Mathematics, vol 5 (1955), pp. 285-309.

/WaSh77/ A.Shamir, W.W.Wadge: Data types as objects, in: Proceedings 4th ICALP, Lecture Notes in Computer Science 52, Springer-Verlag, pp. 465-479.

/Wil81/ R.Wilhelm: Global flow analysis and optimization in the MUG2 compiler generating system, in: Program Flow Analysis: Theory and Applications, S.S.Muchnick, N.D.Jones (eds.), Prentice-Hall 1981.

INDEX